МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ОБРАЗОВАНИЕ

НАЦИОНАЛЬНЫЕ ПРОЕКТЫ РОССИИ

СЕТЬ ЦЕНТРОВ ЦИФРОВОГО ОБРАЗОВАНИЯ ДЕТЕЙ «IT-КУБ»

РЕАЛИЗАЦИЯ ДОПОЛНИТЕЛЬНОЙ ОБЩЕОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ ПО ТЕМАТИЧЕСКОМУ НАПРАВЛЕНИЮ

«ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ JAVA»

С ИСПОЛЬЗОВАНИЕМ ОБОРУДОВАНИЯ ЦЕНТРА ЦИФРОВОГО ОБРАЗОВАНИЯ ДЕТЕЙ «IT-КУБ»

МОСКВА 2021

С. Г. Григорьев

Р. Э. Сабитов

Г. С. Смирнова

Ш. Р. Сабитов

Реализация дополнительной общеобразовательной программы по тематическому направлению «Программирование на языке Java» с использованием оборудования центра цифрового образования детей «IT- куб»

Методическое пособие

под ред. С. Г. Григорьева

Москва, 2021

Введение

Цель и задачи создания центров цифрового образования детей «IT-куб»

Целями и задачами центров цифрового образования детей «IT-куб» являются реализация программ дополнительного образования, проведение мероприятий по тематике современных цифровых технологий и информатики, знакомство детей с технологиями искусственного интеллекта, а также обеспечение просветительской работы по цифровой грамотности и цифровой безопасности.

Задачами центра являются:

- реализация разноуровневых дополнительных общеобразовательных программ для детей;
- разработка и реализация иных программ, в том числе в каникулярный период;
- вовлечение обучающихся и педагогических работников в проектную деятельность;
- организация внеучебной деятельности в каникулярный период;
- разработка и реализация соответствующих образовательных программ, в том числе для лагерей, организованных образовательными организациями в каникулярный период;
- повышение профессионального мастерства педагогических работников центра, реализующих дополнительные общеобразовательные программы.

Нормативная база

Конституция Российской Федерации (принята всенародным голосованием 12.12.1993 с изменениями, одобренными в ходе общероссийского голосования 01.07.2020) — URL: http://www.consultant.ru/document/cons_doc_LAW_28399/ (дата обращения: 10.03.2021)

Федеральный закон от 29.12.2012 № 273-ФЗ (ред. от 31.07.2020) «Об образовании в Российской Федерации» (с изм. и доп., вступ. в силу с 01.09.2020) — URL: http://www.consultant.ru/document/cons_doc_LAW_140174/ (дата обращения: 28.09.2020)

Паспорт национального проекта «Образование» (Утверждён президиумом Совета при Президенте РФ по стратегическому развитию и национальным проектам, протокол от 24.12.2018 N 16) — URL: https://login.consultant.ru link ?req=doc&base=LAW-&n=319308&demo=1 (дата обращения: 10.03.2021)

Государственная программа Российской Федерации «Развитие образования» (Утверждена Постановлением Правительства РФ от 26.12.2017 N 1642 (ред. от 22.02.2021) «Об утверждении государственной программы Российской Федерации «Развитие образования» — URL: http://www.consultant.ru/document/cons_doc_LAW_286474/ (дата обращения: 10.03.2021)

Стратегия развития воспитания в Российской Федерации на период до 2025 года (Утверждена распоряжением Правительства РФ от 29.05.2015 N 996-р «Об утверждении Стратегии развития воспитания в Российской Федерации на период до 2025 года») — URL: http://www.consultant.ru/document/cons_doc_LAW_180402/ — (дата обращения: 10.03.2021)

Профессиональный стандарт «Педагог (педагогическая деятельность в дошкольном, начальном общем, основном общем, среднем общем образовании), (воспитатель, учитель)» (ред. от 16.06.2019 г.) (Приказ Министерства труда и социальной защиты РФ

от 18 октября 2013 г. № 544н, с изменениями, внесенными приказом Министерства труда и соцзащиты РФ от 25 декабря 2014г. № 1115н и от 5 августа 2016г. № 422н) — URL: http://профстандартпедагога.рф — (дата обращения: 10.03.2021)

Профессиональный стандарт «Педагог дополнительного образования детей и взрослых» (Приказ Министерства труда и социальной защиты РФ от 5 мая 2018 г. N 298н «Об утверждении профессионального стандарта «Педагог дополнительного образования детей и взрослых») — URL: https://profstandart.rosmintrud.ru/obshchiy-informatsionnyy-blok/natsionalnyy-reestr-professionalnykh-standartov/reestr-professionalnykh-standartov/index.php?ELEMENT_ID=48583 (дата обращения: 10.03.2021)

Федеральный государственный образовательный стандарт основного общего образования (Утверждён приказом Министерства образования и науки Российской Федерации от 17 декабря 2010 г. N 1897) (ред. 21.12.2020) — URL: https://fgos.ru (дата обращения: 10.03.2021)

Федеральный государственный образовательный стандарт среднего общего образования (Утверждён приказом Министерства образования и науки Российской Федерации от 17 мая 2012 г. N 413) (ред. 11.12.2020) — URL: https://fgos.ru (дата обращения: 10.03.2021)

Методические рекомендации по созданию и функционированию детских технопарков «Кванториум» на базе общеобразовательных организаций (Утверждены распоряжением Министерства просвещения Российской Федерации от 12 января 2021 г. N Р-4) — URL: http://www.consultant.ru/document/cons_doc_LAW_374695/ (дата обращения: 10.03.2021)

Методические рекомендации по созданию и функционированию центров цифрового образования «IT-куб» (Утверждены распоряжением Министерства просвещения Российской Федерации от 12 января 2021 г. N Р-5) — URL: http://www.consultant.ru/document/cons_doc_LAW_374572/ (дата обращения: 10.03.2021)

Методические рекомендации по созданию и функционированию в общеобразовательных организациях, расположенных в сельской местности и малых городах, центров образования естественно-научной и технологической направленностей («Точка роста») — (Утверждены распоряжением Министерства просвещения Российской Федерации от 12 января 2021 г. N Р-6) — URL: http://www.consultant.ru/document/cons_doc_LAW_374694/ (дата обращения: 10.03.2021)

Основные понятия и термины

Справочник

IT-куб — центр образования детей по программам, направленным на ускоренное освоение актуальных и востребованных знаний, навыков и компетенций в сфере информационных технологий.

Язык программирования — формальный язык, представляющий собой набор формальных правил, по которым пишут компьютерные программы.

Java — высокоуровневый кроссплатформенный объектно-ориентированный язык со строгой типизацией.

ООП — сокращение от термина «объектно-ориентированное программирование».

IDE — интегрированная среда разработки.

JDK — Java Development Kit, платформа для разработки на языке Java.

JRE — ядро платформы JDK.

JVM — Java Virtual Machine, виртуальная машина Java, специальная среда для выполнения байт-кода.
IntelliJ IDEA — интегрированная среда разработки программ на Java компании JetBrains.
Компиляция — формирование машинного кода из программного.
Консоль — специальное окно редактора IntelliJ для ввода и вывода данных.
Переменная — область памяти компьютера, имеющая имя и содержащая данные.
Оператор — конструкция языка, определяющая команду (набор команд) языка программирования, задающая выполнение действий.
Класс — ключевое понятие в объектно-ориентированном программировании, шаблон для создания объектов, задающий начальные значения переменных и поведение функций и методов. Базовая структурная единица языка Java.
Условный оператор — оператор, который используется для выбора выполнения той или иной последовательности действий в зависимости от истинности или ложности некоторого условия.
Оператор цикла — оператор, который выполняет одну и ту же последовательность действий несколько раз; количество повторений либо задано, либо зависит от истинности или ложности некоторого условия.
Список — упорядоченная изменяемая последовательность элементов различного типа.
Массив — структура данных, хранящая набор значений (элементов массива), обозначающихся индексом или набором индексов.
Отладчик — специальное средство разработки для проверки корректности программного кода.
Поток управления — способ выполнения процесса, задающий порядок выполнения программного кода.
Синтаксический сахар — упрощённые для удобства и скорости разработки синтаксические конструкции языка программирования.

Структурирование материалов

Содержание обучения может быть представлено следующими разделами.

1. Знакомство со средой программирования IntelliJ. Создание первого проекта.
2. Переменные. Операторы.
3. Ввод данных.
4. Управляющие структуры. Последовательные инструкции. Ветвления.
5. Классы.
6. Статические элементы.
7. Управляющие структуры. Циклы.
8. Массивы.
9. Работа со строками.
10. Списки.
11. Отладка кода.

Для каждого раздела в данном пособии представлены лабораторные работы с необходимым теоретически материалом, заданиями и указанием к их выполнению. Также имеются справочные материалы общей направленности, которые можно использовать при подготовке преподавателей и учащихся к занятиям при выполнении лабораторных работ.

Описание материально-технической базы центров цифрового образования детей «IT-куб»

Для организации работы центра «IT-куб» в распоряжении «Об утверждении методических рекомендаций по созданию и функционированию центров цифрового образования «IT-куб» от 12.02.2021 рекомендуется следующее оборудование лаборатории.

Рабочее место преподавателя	
Ноутбук тип 1	Форм-фактор: ноутбук. Жёсткая, неотключаемая клавиатура. Русская раскладка клавиатуры. Диагональ экрана: не менее 15,6 дюймов. Разрешение экрана: не менее 1920 × 1080 пикселей. Количество ядер процессора: не менее 4. Количество потоков: не менее 8. Базовая тактовая частота процессора: не менее 1 ГГц. Максимальная тактовая частота процессора: не менее 2,5 ГГц. Кэш-память процессора: не менее 6 Мбайт. Объём установленной оперативной памяти: не менее 8 Гбайт. Объём поддерживаемой оперативной памяти (для возможности расширения): не менее 24 Гбайт. Объём накопителя SSD: не менее 240 Гбайт. Время автономной работы от батареи: не менее 6 часов. Вес ноутбука с установленным аккумулятором: не более 1,8 кг. Внешний интерфейс USB стандарта не ниже 3.0: не менее трёх свободных. Внешний интерфейс LAN (использование переходников не предусмотрено). Наличие модулей и интерфейсов (использование переходников не предусмотрено): VGA, HDMI. Беспроводная связь Wi-Fi: наличие с поддержкой стандарта IEEE 802.11n или современнее. Веб-камера. Манипулятор «мышь». Предустановленная операционная система с графическим пользовательским интерфейсом, обеспечивающая работу распространённых образовательных и общесистемных приложений
Веб-камера	Микрофон: наличие, автоматическая фокусировка: наличие
МФУ (принтер, сканер, копир)	Набор функций: принтер/сканер/копир. СНПЧ в составе устройства или СНПЧ, совместимая с МФУ в комплекте поставки. Печать цветных изображений. Максимальный формат печати: А3, с максимальным разрешением печати не хуже 4800 × 1200 dpi. Скорость печати: не менее 15 стр/мин. Функция автоматической двусторонней печати. Функция печати без полей. Функция беспроводного подключения, как минимум WiFi и AirPrint. Дисплей для отображения информации. Поддержка ОС Windows, macOS, iOS, Android. Интерфейсы подключения USB, RJ45

Продолжение таблицы

Рабочее место обучающегося	
Ноутбук тип 2	Форм-фактор: ноутбук. Жёсткая, неотключаемая клавиатура. Русская раскладка клавиатуры. Диагональ экрана: не менее 15,6 дюймов. Разрешение экрана: не менее 1920 × 1080 пикселей. Количество ядер процессора: не менее 4. Количество потоков: не менее 8. Базовая тактовая частота процессора: не менее 1 ГГц. Максимальная тактовая частота процессора: не менее 2,5 ГГц. Кэш-память процессора: не менее 6 Мбайт. Объём установленной оперативной памяти: не менее 8 Гбайт. Объём поддерживаемой оперативной памяти (для возможности расширения): не менее 24 Гбайт. Объём накопителя SSD: не менее 240 Гбайт. Время автономной работы от батареи: не менее 6 часов. Вес ноутбука с установленным аккумулятором: не более 1,8 кг. Внешний интерфейс USB стандарта не ниже 3.0: не менее трёх свободных. Внешний интерфейс LAN (использование переходников не предусмотрено). Наличие модулей и интерфейсов (использование переходников не предусмотрено): VGA, HDMI. Беспроводная связь Wi-Fi: наличие с поддержкой стандарта IEEE 802.11n или современнее. Веб-камера. Манипулятор «мышь». Предустановленная операционная система с графическим пользовательским интерфейсом, обеспечивающая работу распространённых образовательных и общесистемных приложений
Наушники	Тип: полноразмерные
Презентационное оборудование	
Моноблочное интерактивное устройство	Интерактивный моноблочный дисплей, Диагональ экрана: не менее 65 дюймов, Разрешение экрана: не менее 3840 × 2160 пикселей. Встроенная акустическая система. Количество одновременно распознаваемых касаний сенсорным экраном: не менее 20 касаний. Высота срабатывания сенсора экрана: не более 3 мм от поверхности экрана. Встроенные функции распознавания объектов касания (палец или безбатарейный стилус). Количество поддерживаемых безбатарейных стилусов одновременно: не менее 2 шт. Возможность использования ладони в качестве инструмента стирания либо игнорирования касаний экрана ладонью. Интегрированный датчик освещённости для автоматической коррекции яркости подсветки. Наличие функции графического комментирования поверх произвольного изображения, в том числе от физически подключённого источника видеосигнала.

Продолжение таблицы

	Интегрированные функции вывода изображений с экранов мобильных устройств (на платформе распространённых ОС), а также с возможностью интерактивного взаимодействия (управления) с устройством-источником. Интегрированный в пользовательский интерфейс функционал просмотра и работы с файлами основных форматов с USB-накопителей или сетевого сервера. Поддержка встроенными средствами дистанционного управления рабочих параметров устройства через внешние системы. Предустановленная операционная система с графическим пользовательским интерфейсом, обеспечивающая работу распространённых образовательных и общесистемных приложений. Интегрированные средства, обеспечивающие следующий функционал: создание многостраничных уроков с использованием медиаконтента различных форматов, создание надписей и комментариев поверх запущенных приложений, распознавание фигур и рукописного текста (русский, английский языки), наличие инструментов рисования геометрических фигур и линий, встроенные функции: генератор случайных чисел, калькулятор, экранная клавиатура, таймер, редактор математических формул, электронные математические инструменты: циркуль, угольник, линейка, транспортир, режим «белой доски» с возможностью создания заметок, рисования, работы с таблицами и графиками, импорт файлов форматов: PDF, PPT
Напольная мобильная стойка для интерактивных досок или универсальное настенное крепление	Совместимость с моноблочным интерактивным устройством. Максимальный вес, выдерживаемый креплением: не менее 60 кг
Дополнительное оборудование	
Доска магнитно-маркерная настенная	Тип: полимерная, сухостираемая
Флипчарт магнитно-маркерный на треноге	Размер рабочей области: не менее 700 × 1000 мм. Тип опоры: тренога
Комплект кабелей и переходников	Кабели, переходники для подключения и коммутации оборудования. Сетевой удлинитель для подключения оборудования к сети электропитания и др. (по выбору)
Учебная и методическая литература	Для реализации образовательных программ
Комплект комплектующих и расходных материалов	Для реализации образовательных программ

Окончание таблицы

Мебель	
Комплект мебели	Учебная мебель: столы, для всех учеников, стулья/кресла для всех учеников, пуфы. Мебель для педагога: стол, стул (кресло). Системы хранения: тумбы, шкафы, стеллажи (по выбору)

В центре «IT-куб» действуют несколько лабораторий (рис. 1), в том числе лаборатория для осуществления направления «Программирование на языке Java».

Рис. 1. Лаборатория в «IT-Куб»

Лаборатория оборудована ноутбуками Asus, интерактивной доской, маркерной доской, МФУ.

На данном оборудовании могут выполняться лабораторные работы по учебному курсу «Программирование на языке Java», проводиться открытые занятия, защита проектов и т. д. С использованием презентационного оборудования преподаватели объясняют новый материал, приводят примеры работы программ и т. д.

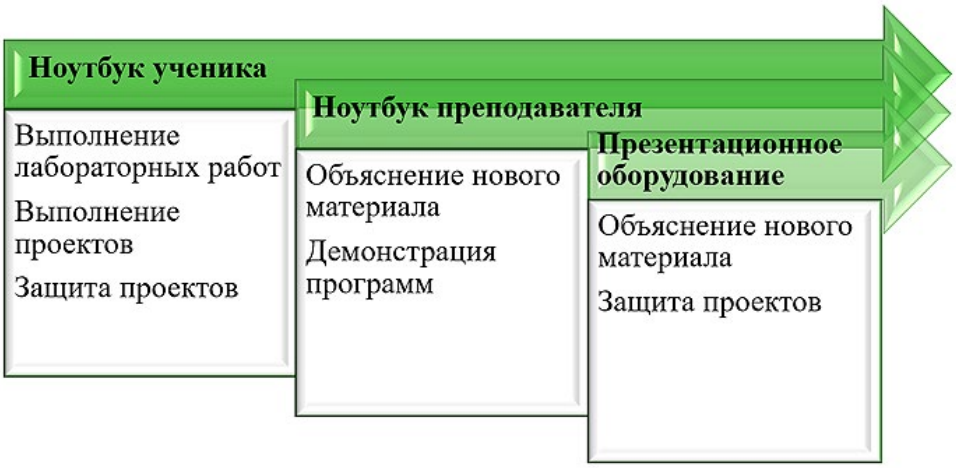

Рис. 2. Использование оборудования IT-куб при организации занятий по курсу «Программирование на языке Java»

Справочные материалы

Платформа JDK

Поскольку код на языке Java для выполнения требует специальную среду в виде виртуальной машины Java (JVM), то необходимо рассмотреть материал, касающийся платформы JDK и её компонентов.

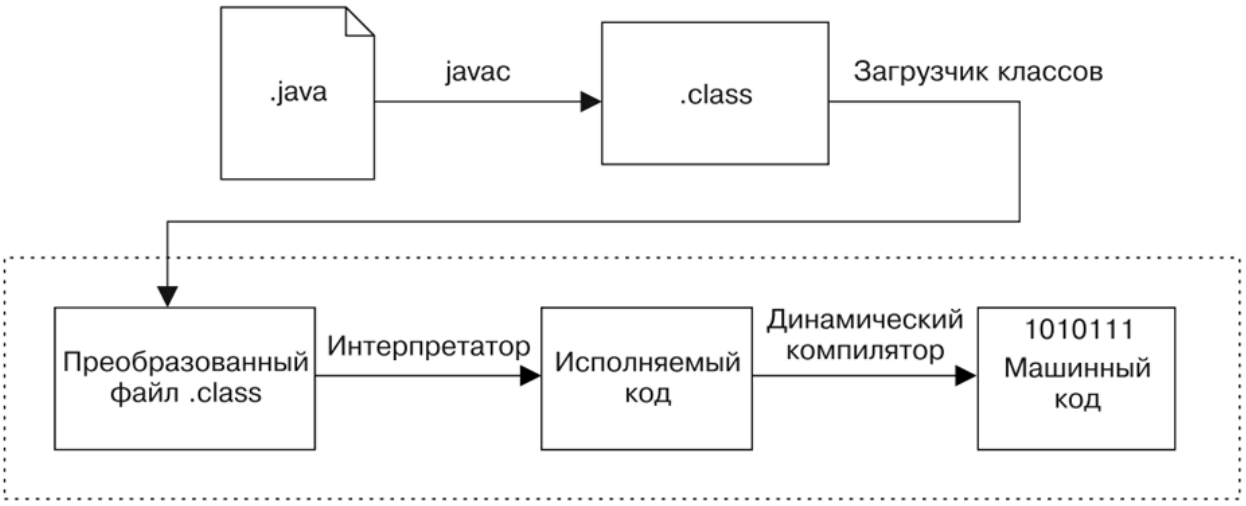

Рис. 3. Схема преобразования Java-кода из программного в машинный код

JVM (Java Virtual Machine) — виртуальная машина Java. Она обеспечивает среду для исполнения и управления байт-кода Java. Виртуальная машина выполняет множество сложных функций — от распределения памяти, управления жизненным циклом объектов и оптимизации памяти приложений до загрузки классов и управления системными ресурсами.

Важно!

Именно благодаря JVM, которая доступна для многих технических систем и аппаратных платформ, возможно столь обширное использование языка Java. Эти функции в своё время и явились причиной взрывной популярности Java. Следует отметить, что в среде JVM может быть исполнен не только Java-код, но и код на многих других языках: Ada, Scala, Groovy, Kotlin и т. д.

JRE (Java Runtime Environment) — совокупность JVM и набора базовых библиотек. В JRE не входят компилятор и ряд других средств, но она содержит минимальный объём Java-библиотек, необходимый для программирования.

JDK (Java Development Kit) — набор инструментов для разработки на языке Java. JDK — содержит JRE и дополнительные библиотеки, компилятор, различные компоненты, примеры, документацию.

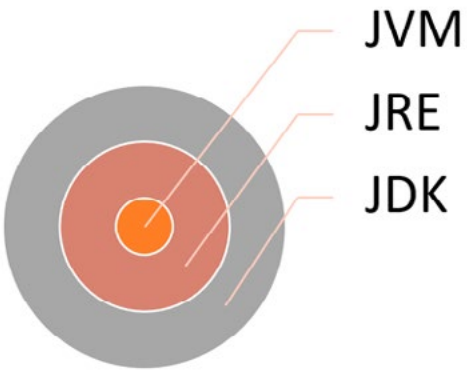

Рис. 4. Структура Java Development Kit

Разработка первой версии JDK началась ещё в 1990 году. Актуальной версией сейчас является JDK 16.0. При разработке в каждую последующую версию вносились нововведения, дополнительные улучшения синтаксиса, JVM. С каждой новой версией JDK в язык Java добавлялось всё больше так называемого «синтаксического сахара», т. е. конструкций языка, упрощающих и ускоряющих написание программного кода.

Однако не всегда «синтаксический сахар» является полезным. Это дискуссионный вопрос, так как в каждом нововведении можно найти как плюсы, так и минусы.

Последней версией JDK, внёсшей самые значительные изменения в процесс разработки программ на Java, является JDK 8.0 Octopus, выпущенная в 2014 году. Особенностью этой версии является появление лямбда-выражения, присущего функциональной парадигме программирования. При этом Java остался языком объектно-ориентированной парадигмы, «под капотом» элементов функционального программирования находится тот же самый объектно-ориентированный код, который с помощью «синтаксического сахара» представляется в функциональном стиле.

Важно!

Одной из важнейших стадий в эволюции языка Java было введение обратной совместимости кода и проектов (legacy code), чтобы при появлении новых библиотек Java, при развитии JDK код, написанный с использованием предыдущих версий JDK, продолжал поддерживаться, и не приходилось переписывать большие проекты для поддержки новой версией.

Это одна из причин медленного развития Java в определённые периоды: синтаксис отставал от трендов в программировании, добавлялось мало «синтаксического сахара». Но в то же время это означает надёжность языка, особенно при поддержке старых проектов, что крайне важно для бесперебойной работы крупных технических систем на базе этого языка. Можно сказать, что код Java достаточно консервативен. До сих пор во многих отраслях эксплуатируется оборудование, работающее на основе программ, написанных ещё в 90-е годы.

В состав JDK не входит интегрированная среда разработки на Java, поэтому разработчик, использующий только JDK, вынужден использовать внешний текстовый редактор (например, «Блокнот») и компилировать свои программы, используя командную строку (например, с помощью Windows-утилиты «cmd», вызывая команды javac, java и т. д.).

Установка JDK

Несмотря на то, что компьютеры в лабораториях «IT-Куб» уже оснащены необходимым программным обеспечением, нужно осуществить процесс установки и настройки среды программирования на Java совместно с обучающимися, чтобы они смогли самостоятельно установить и настроить ПО на своих домашних компьютерах или ноутбуках с целью выполнения домашних заданий и проработки курса.

Инструкция по установке JDK

Дистрибутив OpenJDK можно скачать по ссылке https://www.azul.com/downloads/zulu-community/?architecture=x86-64-bit&package=jdk.

Выберите свою операционную систему и её архитектуру (32 или 64-битную).

Рис. 5. Выбор типа загрузчика

В диалоговом окне выберите временную папку для загрузки и сохранения дистрибутива.

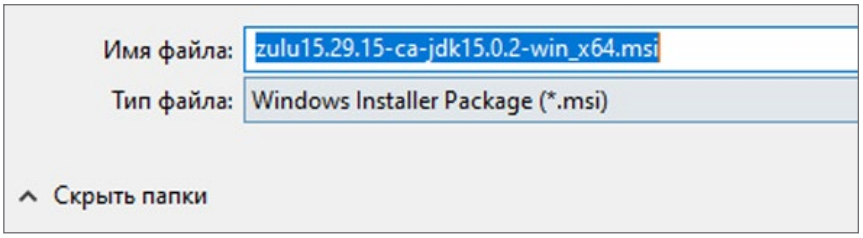

Рис. 6. Диалоговое окно выбора папки для сохранения дистрибутива

Нажмите кнопку *Сохранить* и дождитесь завершения загрузки установочного файла OpenJDK.

После завершения загрузки перейдите к скачанному файлу и запустите его, нажав клавишу *Enter* или с помощью двойного щелчка мыши.

В появившемся диалоговом окне установщика нажмите кнопку *Next*.

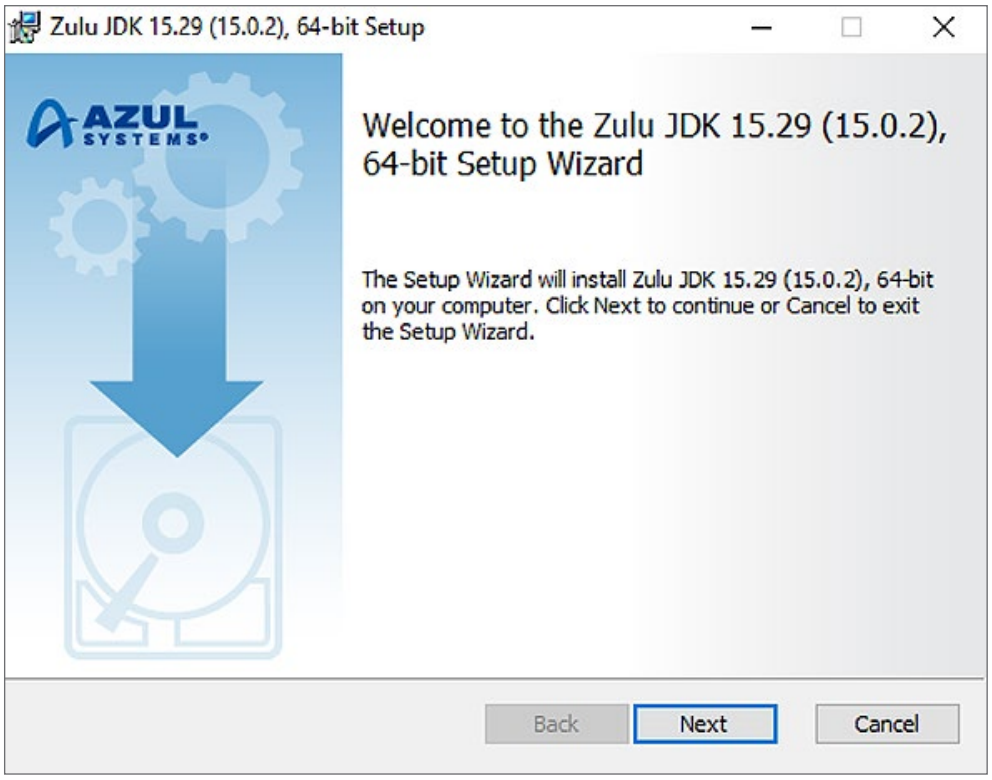

Рис. 7. Стартовое диалоговое окно установщика

В следующем окне выбора папки для установки рекомендуется оставить путь по умолчанию. Снова нажмите кнопку *Next*.

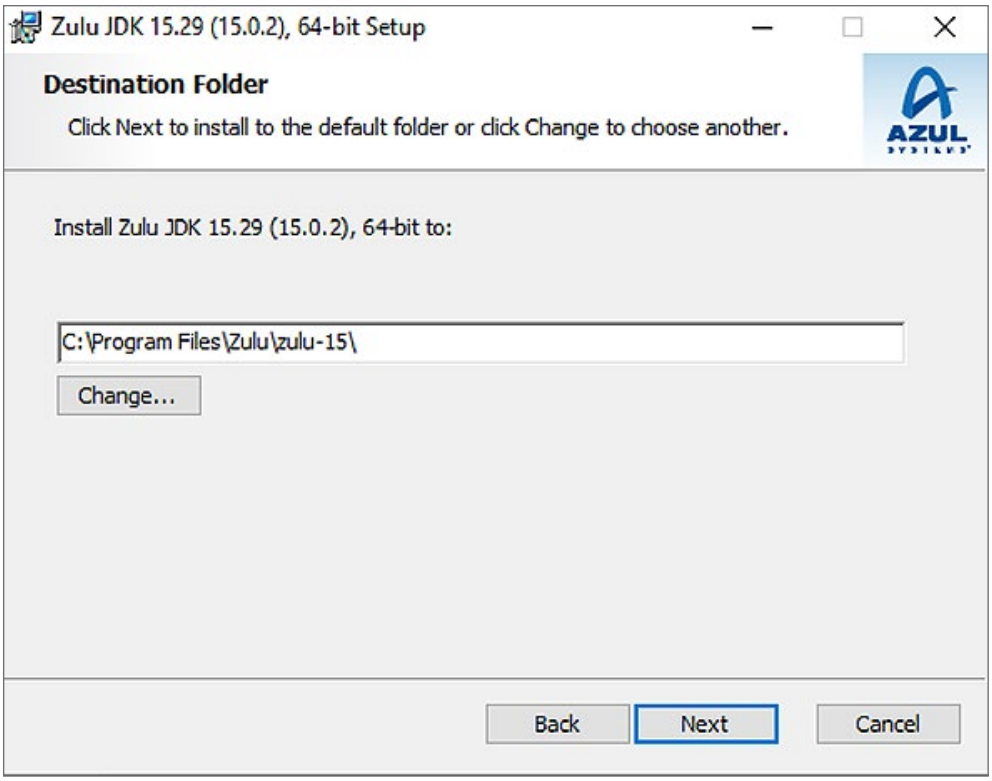

Рис. 8. Окно выбора папки для установки

В окне завершения установки нажмите кнопку *Install*. Дождитесь процесса завершения установки.

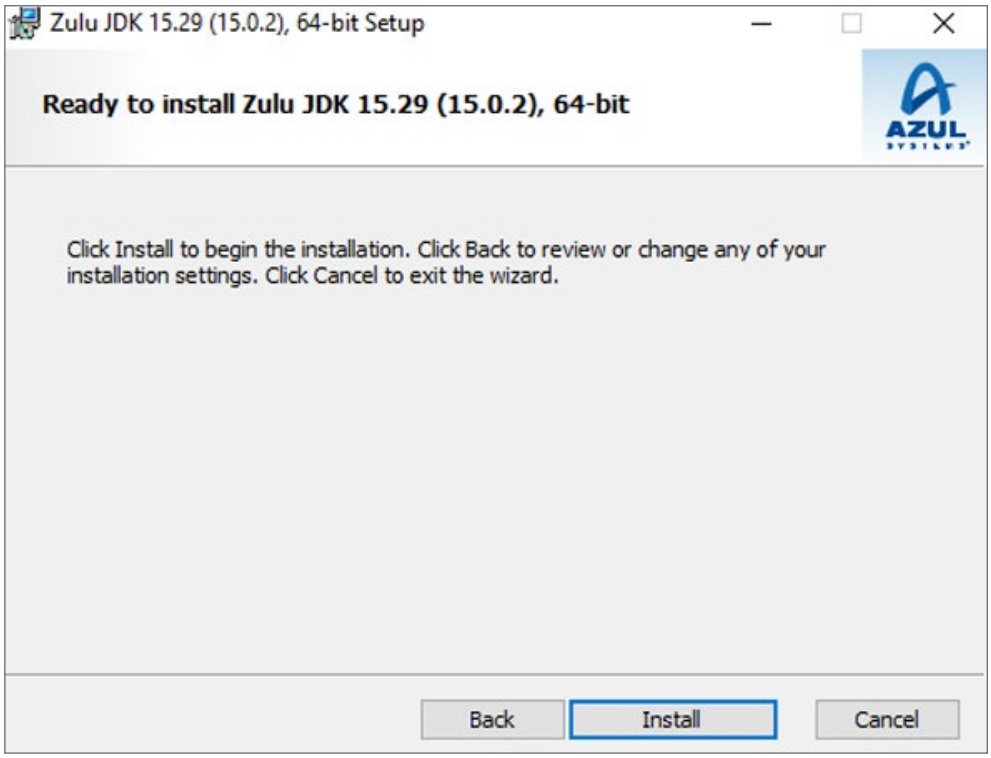

Рис. 9. Окно завершения установки

В финальном окне установки нажмите кнопку *Finish*.

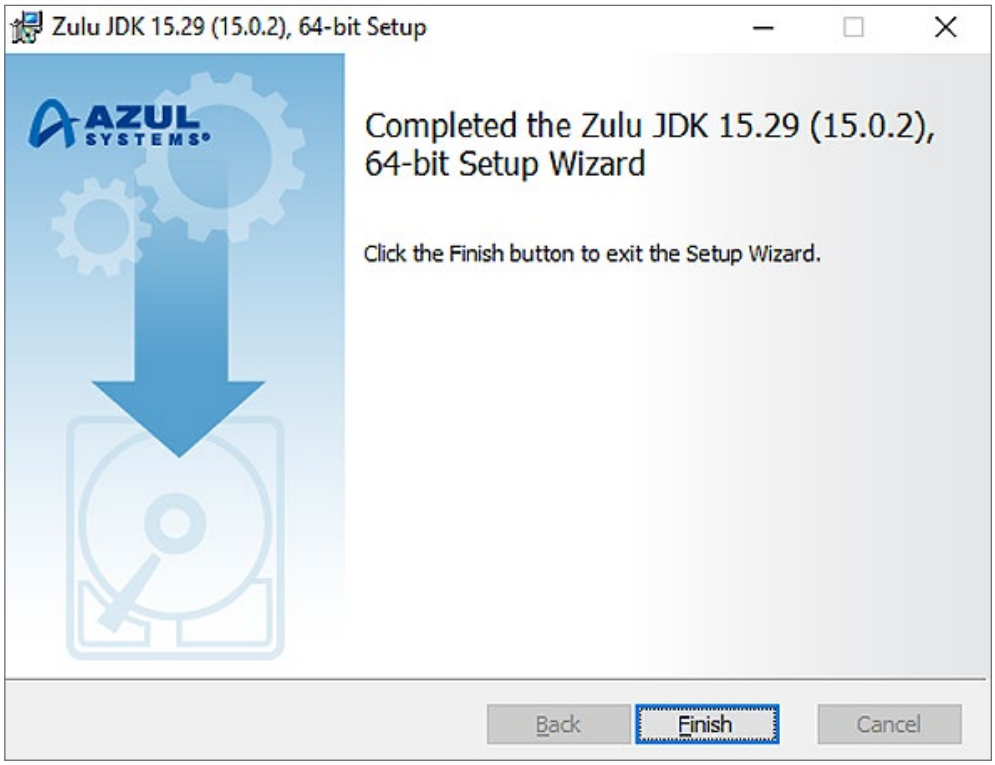

Рис. 10. Финальное окно установки

Теперь версия Zulu JDK доступна к использованию.

Установка среды IntelliJ IDEA

Для установки IntelliJ Idea для Windows необходимо зайти на сайт компании JetBrains и скачать версию «Community»: https://www.jetbrains.com/ru-ru/idea/download/#section=windows.

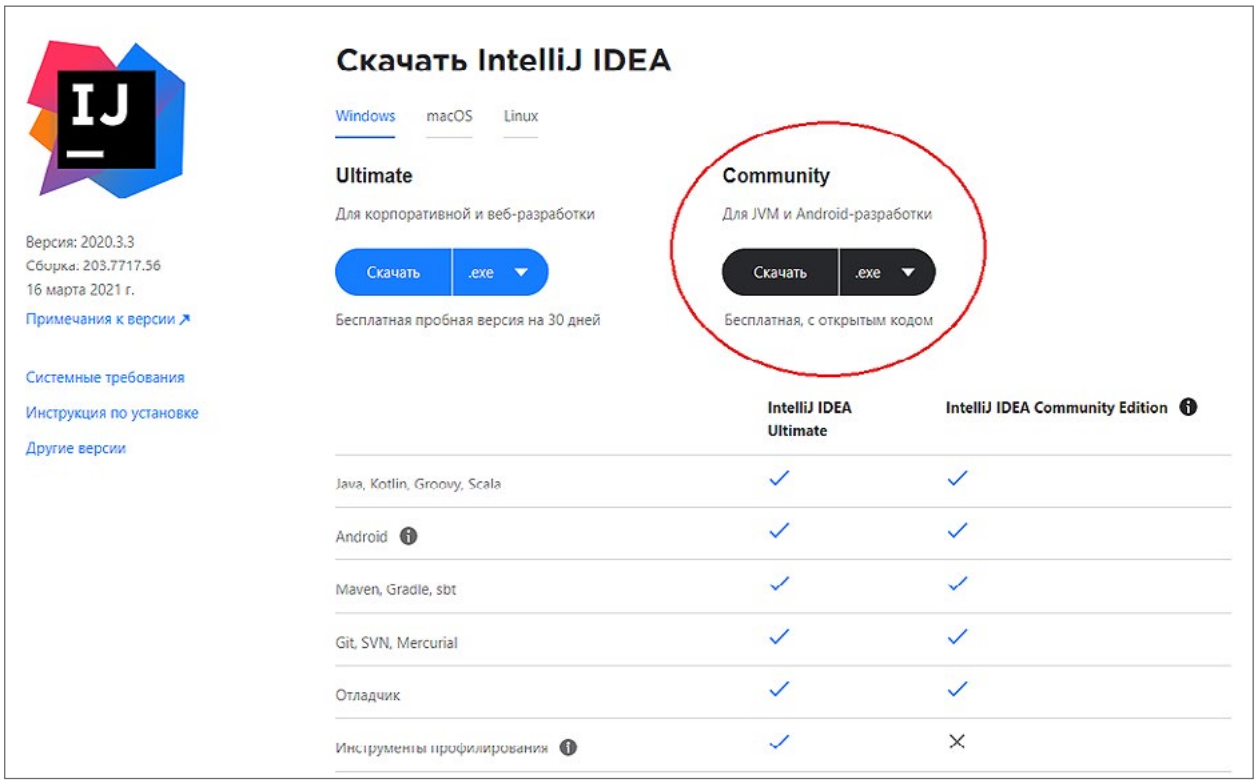

Рис. 11. Страница скачивания дистрибутива IntelliJ IDEA

Нажмите кнопку *Скачать*.

В диалоговом окне выберите временную папку для загрузки и сохранения дистрибутива. Нажмите кнопку Сохранить и дождитесь завершения загрузки установочного файла IntelliJ Community Edition.

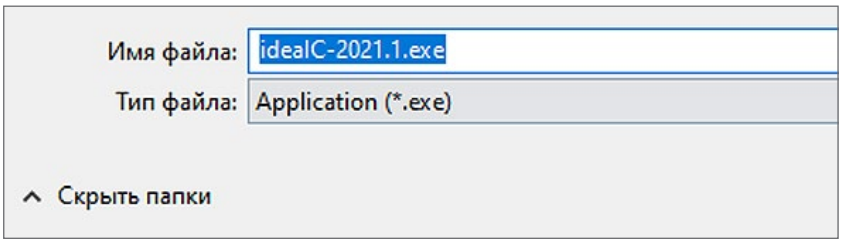

Рис. 12. Диалоговое окно выбора папки для сохранения дистрибутива

После завершения загрузки перейдите к скачанному файлу и запустите его, нажав клавишу *Enter* или с помощью двойного щелчка мыши.

В появившемся диалоговом окне установщика нажмите кнопку *Next*.

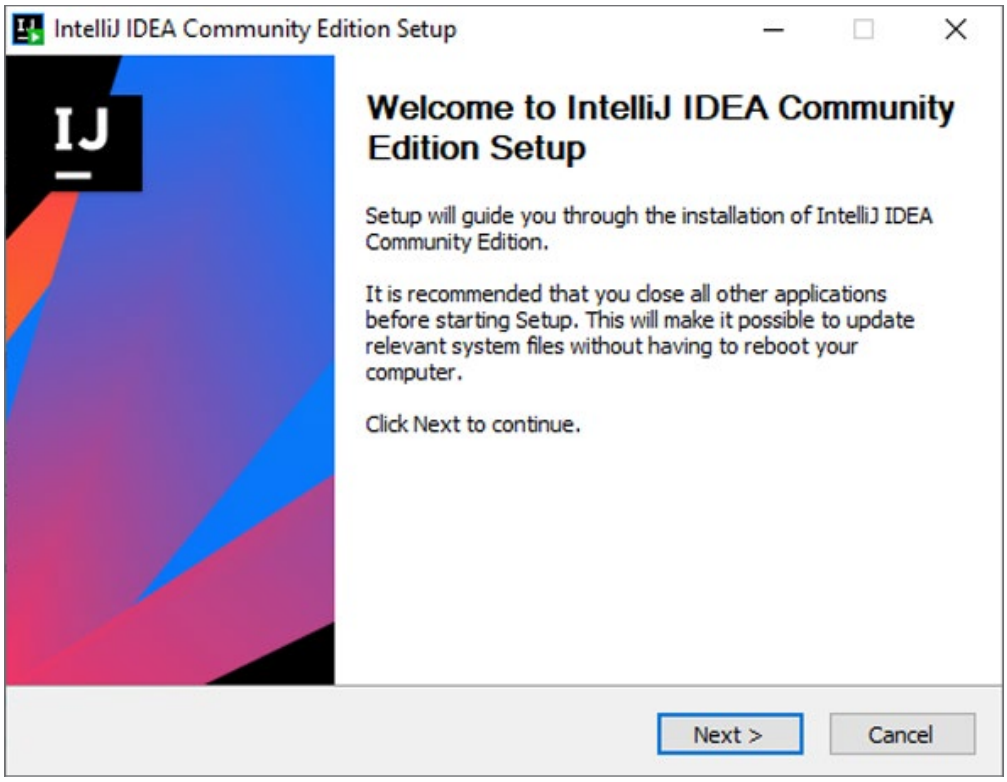

Рис. 13. Стартовое диалоговое окно установщика

В окне выбора папки для установки рекомендуется оставить путь по умолчанию. Фраза «Space required: 1.6 GB» (цифра может отличаться в зависимости от версии программы) означает, что для установки требуется 1.6 Гб свободного пространства на жестком диске. Нажмите кнопку *Next*.

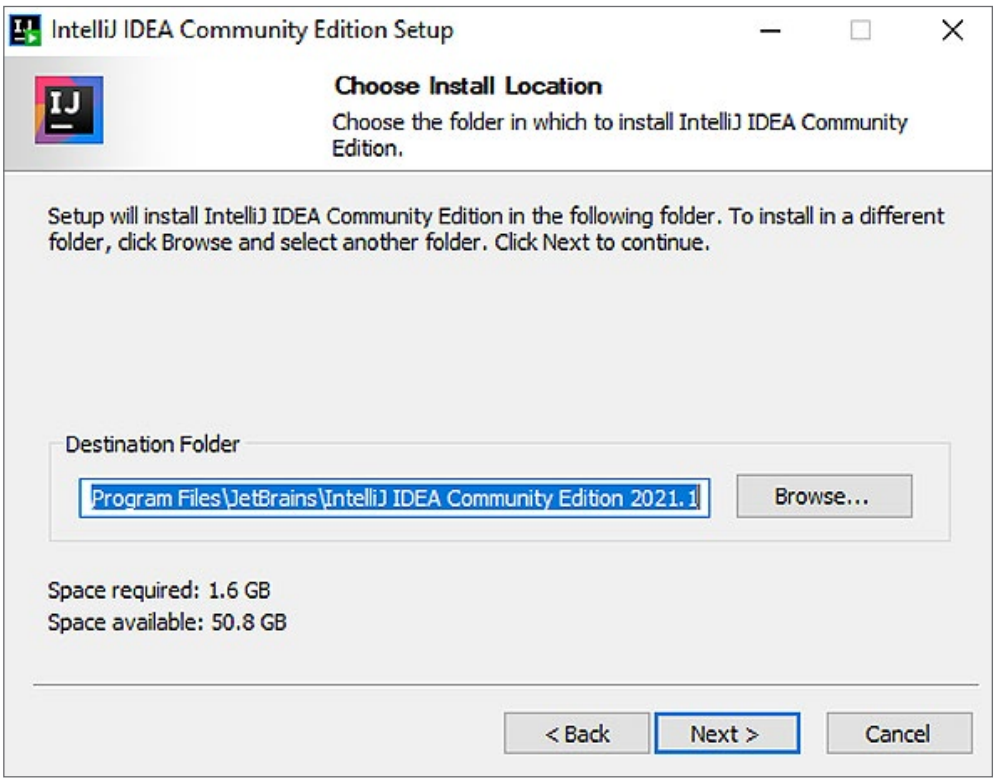

Рис. 14. Окно выбора папки для установки

В окне выбора параметров установки отметьте галочками «64-bit launcher» или «32-bit launcher» в зависимости от архитектуры системы и «.java».

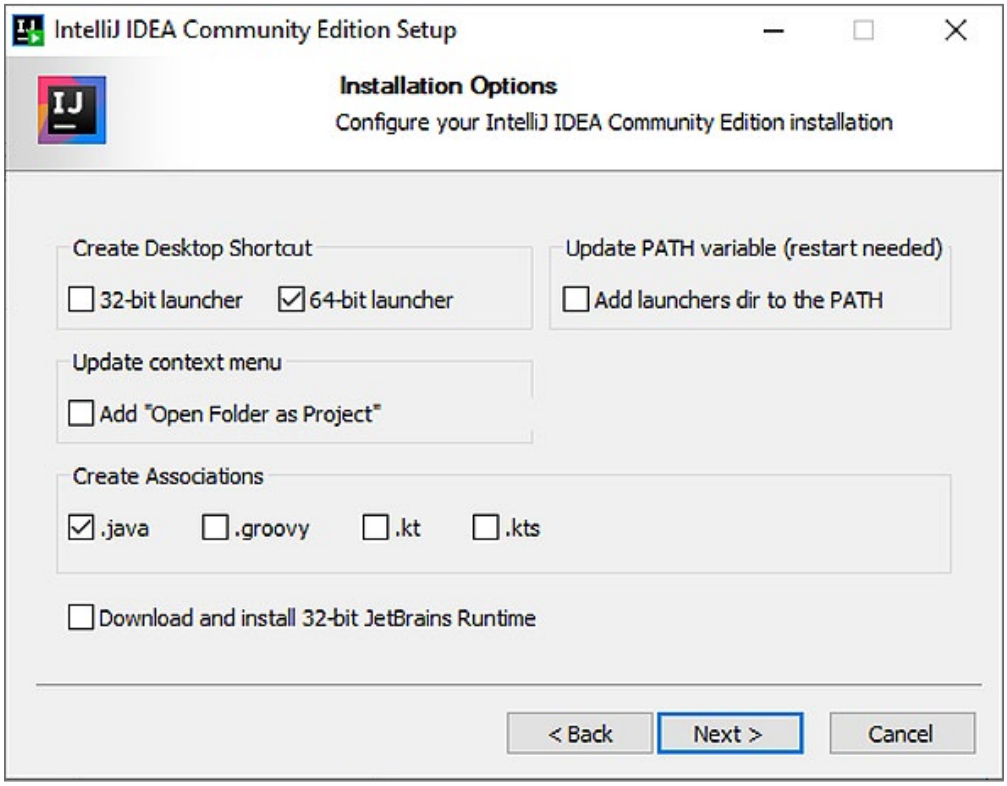

Рис. 15. Параметры установки

В окне выбора папки для ярлыков нажмите кнопку *Install*. Подождите завершения установки. После завершения процесса установки нажмите кнопку *Finish*.

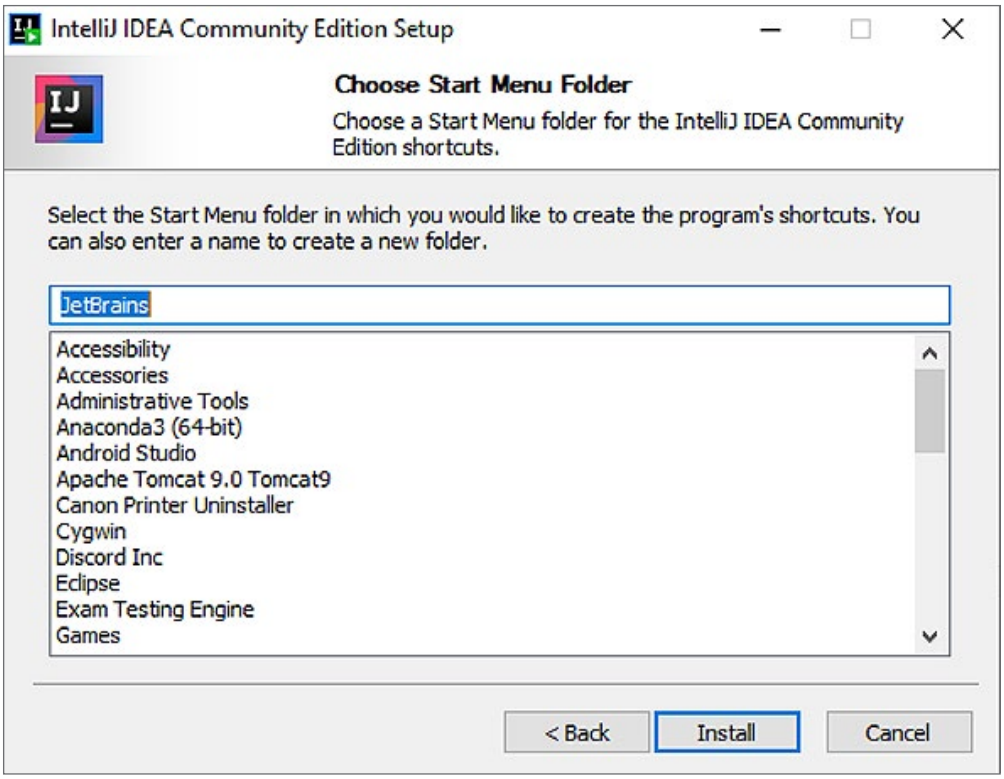

Рис. 16. Окно выбора папки для создания ярлыков IntelliJ

Java SE

В данном курсе рассматриваются основы языка Java, поэтому речь будет идти о технологии Java SE (Java Standard Edition). Кроме Java SE, существует также технология Java EE (Java Enterprise Edition). Java SE — это набор библиотек, фреймворков и технологий на базе языка Java, который изучается в данном пособии. А Java EE — набор компонентов, библиотек, фреймворков, в том числе и для промышленной разработки ПО на базе Java. Java EE содержит инструменты для создания web-приложений, масштабируемых компонентов, для управления безопасностью и транзакциями, многопоточностью и т. д. Java EE можно рассматривать как надстройку над Java SE и языком Java.

Примерная рабочая программа дополнительного образования для организации работы по тематическому направлению «Программирование на языке Java» деятельности центра цифрового образования детей «IT-куб»

Цель освоения программы

Важно!

Язык Java является одним из популярнейших современных высокоуровневых языков программирования. Характерной особенностью этого языка является кросс-платформенность и интегрируемость со многими техническими системами.

Язык Java используется во многих областях — от серверных и десктопных приложений до веб-разработки, IoT, финансовых систем, мобильной разработки и т. д. На сегодняшний день более трёх миллиардов устройств в мире используют Java.

Последние 20 лет Java стабильно занимает первые и вторые места в мировом рейтинге языков программирования TIOBE. Язык Java официально увидел свет летом 1995 года. В этот год компания Sun Microsystems выпустила первую версию Java 1.0., хотя проекты, на которые опирались разработчики языка при его создании, начались ещё в 1991 году. Вторая версия языка Java 1.1 была выпущена уже в 1997 году. На сегодняшний день уже выпущена 16-я версия. Считается, что название «Java» связано с одноимённым сортом кофе, выращиваемым на острове Ява в Индонезии.

Язык Java является сильно типизированным объектно-ориентированным языком, поэтому невозможно изучать программирование на Java, не рассматривая объектно-ориентированные средства языка. В данном методическом пособии рассматриваются как типовые, процедурно-алгоритмические аспекты языка программирования, так и объектно-ориентированные. Целью является приобретение навыков владения не только процедурным, но и объектно-ориентированным кодом на языке Java, развитие не только алгоритмического, но и объектно-ориентированного стиля мышления. Таким образом при дальнейшем изучении программирования у учащихся будет меньше сложностей при освоении объектно-ориентированных языков высокого уровня, играющих очень важную роль в современном программировании, особенно в программировании комплексных динамических и эволюционирующих систем и программных комплексов.

Целями учебного курса «Программирование на языке Java» являются:

- освоение базового синтаксиса и возможностей языка Java для получения навыков создания простых приложений;
- получение навыков оперирования программным кодом с учётом специфики данного языка;
- развитие навыков анализа кода, совершенствование алгоритмического мышления и творческих способностей учащихся;
- освоение базовых объектно-ориентированных возможностей языка;
- обеспечение базы для дальнейшего более глубокого освоения либо языка Java и сопутствующих ему фреймворков и технологий, либо других современных объектно-ориентированных высокоуровневых языков.

Для достижения поставленных целей планируется выполнение следующих задач.

Образовательные:
- формирование представления о структуре и функционировании стандартной платформы Java;
- формирование умения использовать инструменты интегрированный среды разработки IntelliJ IDEA Community Edition для решения поставленных задач;
- формирование представления о базовом синтаксисе Java, необходимом для реализации процедурного кода и решения типовых алгоритмических задач;
- формирование умения и навыка построения различных видов алгоритмов (линейных, разветвляющихся, циклических) в среде IntelliJ IDEA для решения поставленных задач;
- формирование умения использовать ряд базовых средств языка Java для решения типовых прикладных задач;
- формирование представления об основах объектно-ориентированной парадигмы и основах синтаксиса Java, необходимого для работы в рамках данной парадигмы;
- формирование умения и навыка применения объектно-ориентированного подхода в языке Java для решения некоторых задач;
- формирование ключевых компетенций проектной и исследовательской деятельности.

Развивающие:
- развитие алгоритмического и логического мышления;
- развитие навыков постановки задачи, выделения основных объектов, математического моделирования;
- развитие умения поиска необходимой учебной информации;
- формирование мотивации к изучению программирования.

Воспитательные:
- воспитание умения работать индивидуально и в группе для решения поставленной задачи;
- воспитание трудолюбия, упорства, желания добиваться поставленной цели;
- воспитание информационной культуры.

Программа рассчитана на учащихся в возрасте от 11 до 15 лет, не требует предварительных знаний и входного тестирования.
Занятия проводятся в группах до 12 человек, продолжительность одного занятия — 45 минут, общая продолжительность курса — 36 часов.

Планируемые результаты освоения программы

Личностные:
- формирование умений и развитие навыков самостоятельной деятельности;
- формирование умения работать в команде;
- формирование коммуникативных навыков;
- формирование навыков анализа и самоанализа;
- формирование эстетического отношения к языкам программирования, осознание их выразительных возможностей;
- формирование целеустремлённости и усидчивости в процессе творческой, исследовательской работы и учебной деятельности.

Предметные:
- формирование основных приёмов работы в среде IntelliJ;
- формирование навыка работы с базовыми языковыми конструкциями языка Java;
- формирование представления об основных алгоритмических конструкциях: линейная, ветвление, цикл;
- формирование навыка использования основных приёмов работы с массивами и динамическими списками;
- формирование навыков отладки программного кода;
- формирование навыка использования основных приёмов работы со строковыми данными;
- формирование представления о понятиях «класс» и «объект»;
- формирование основных приёмов составления программ на языке Java, используя процедурный и объектно-ориентированный подходы;
- формирование алгоритмического и объектно-ориентированного стилей мышления.

Метапредметные:
- формирование умения ориентировки в системе знаний;
- формирование умения выбора наиболее эффективных способов решения задач на компьютере в зависимости от конкретных условий;
- формирование приёмов проектной деятельности, включая умения видеть проблему, формулировать тему и цель проекта, составлять план своей деятельности, осуществлять действия по реализации плана, результат своей деятельности соотносить с целью, классифицировать, наблюдать, проводить эксперименты, делать выводы и заключения, доказывать, защищать свои идеи, оценивать результаты своей работы;
- формирование умения распределения времени;
- формирование умений успешной самопрезентации.

Тематическое планирование

№ п/п	Тема	Содержание	Целевая установка	Кол-во часов	Основные виды деятельности обучающихся на уроке/ внеурочном занятии	Использование оборудования
1.	Знакомство со средой IntelliJ. Создание первого проекта	Установка платформы JDK. Установка среды IntelliJ IDEA Community Edition. Знакомство со средой. Создание простого проекта «Hello, World!»	Ознакомиться с инструментами среды IntelliJ. Создать первый проект «Hello, world!»	2	Наблюдение за работой учителя, самостоятельная работа в IntelliJ, ответы на контрольные вопросы, участие в дискуссии. Выполнение лабораторной работы	Компьютер, проектор, интерактивная доска
2.	Переменные. Операторы	Переменные. Примитивы. Операторы. Основы написания кода на языке Java	Ознакомиться с основами языка Java. Рассмотреть основные типы данных, операторы и ключевые слова	3	Наблюдение за работой учителя, самостоятельная работа в IntelliJ, ответы на контрольные вопросы, участие в дискуссии. Выполнение лабораторной работы	Компьютер, проектор, интерактивная доска
3.	Ввод данных	Работа с классом `Scanner`. Методы `next()`, `hasNext()`	Ознакомиться с инструментов ввода данных через консоль	3	Наблюдение за работой учителя, самостоятельная работа в IntelliJ, ответы на контрольные вопросы, участие в дискуссии. Выполнение лабораторной работы	Компьютер, проектор, интерактивная доска
4.	Классы. Статические элементы	Понятие классов и объектов. Методы. Области видимости и модификаторы доступа. Параметры. Конструкторы. Статические поля и методы	Понять, что такое класс и объект. Ознакомиться с возможностями классов. Рассмотреть переменные объектного	5	Наблюдение за работой учителя, самостоятельная работа в IntelliJ, ответы на контрольные вопросы, участие в дискуссии. Выполнение лабораторной работы	Компьютер, проектор, интерактивная доска

СЕТЬ ЦЕНТРОВ ЦИФРОВОГО ОБРАЗОВАНИЯ ДЕТЕЙ «IT-КУБ»

Продолжение

№ п/п	Тема	Содержание	Целевая установка	Кол-во часов	Основные виды деятельности обучающихся на уроке/ внеурочном занятии	Использование оборудования
			типа. Ознакомиться с возможностями применения статических элементов класса			
5.	Управляющие структуры	Последовательный код, ветвления, циклы. Условные операторы и конструкции. Логические операции	Получить навыки составления алгоритмов с использованием управляющих структур языка Java. Ознакомиться с ветвлениями и условными алгоритмами, операторами. Научиться составлять условия. Научиться работать с циклами в языке Java	6	Наблюдение за работой учителя, самостоятельная работа в IntelliJ, ответы на контрольные вопросы, участие в дискуссии. Выполнение лабораторной работы	Компьютер, проектор, интерактивная доска
6.	Массивы	Одномерные и двумерные массивы	Ознакомиться со структурой данных «массив», способами работы с массивами и их применением	3	Наблюдение за работой учителя, самостоятельная работа в IntelliJ, ответы на контрольные вопросы, участие в дискуссии. Выполнение лабораторной работы	Компьютер, проектор, интерактивная доска
7.	Списки	Динамический список — класс `ArrayList`. Класс как структура данных	Ознакомиться с динамическими списками. Сравнить списки с массивами. Понять, что такое параметризованный список	2	Наблюдение за работой учителя, самостоятельная работа в IntelliJ, ответы на контрольные вопросы, участие в дис-	Компьютер, проектор, интерактивная доска

В содержание

Окончание

					куссии. Выполнение лабораторной работы	
8.	Работа со строками	Строковые данные. Классы `String` и `StringBuffer`	Ознакомиться с методами манипулирования строковыми данными	2	Наблюдение за работой учителя, самостоятельная работа в IntelliJ, ответы на контрольные вопросы, участие в дискуссии. Выполнение лабораторной работы	
9.	Контрольная работа № 1	Решение задач	Проверка полученных навыков по темам «Управляющие структуры»	1	Самостоятельное выполнение контрольных заданий	Компьютер
10.	Отладка кода	Отладка кода средствами среды IntelliJ	Ознакомиться с функциональными возможностями отладчика IntelliJ. Научиться производить отладку кода и вести поиск ошибок	2	Наблюдение за работой учителя, самостоятельная работа в IntelliJ, ответы на контрольные вопросы, участие в дискуссии. Выполнение лабораторной работы	Компьютер, проектор, интерактивная доска
11.	Контрольная работа № 2	Решение задач	Проверка полученных навыков по темам «Классы», «Списки»	1	Самостоятельное выполнение контрольных заданий	Компьютер
12.	Индивидуальное задание	Разработка индивидуального или группового проекта	Создание индивидуального проекта в среде IntelliJ	6	Самостоятельная индивидуальная или групповая проектная деятельность	Компьютер, проектор, интерактивная доска
13.	Итоги	Защита индивидуальных или групповых проектов, подведение итогов курса	Защита проекта	1	Самостоятельная индивидуальная или групповая проектная деятельность	Компьютер, проектор, интерактивная доска
	Итого			36		

В содержание

СЕТЬ ЦЕНТРОВ ЦИФРОВОГО ОБРАЗОВАНИЯ ДЕТЕЙ «IT-КУБ»

Содержание и форма организации учебных занятий

Планы учебных занятий

1. Знакомство со средой IntelliJ. Создание первого проекта.

Рекомендуемое количество часов на данную тему — 2 часа.

Планируемые результаты:

предметные: получение навыков работы в среде IntelliJ, освоение основных инструментов среды; получение умений установки платформы JDK и среды IntelliJ.

метапредметные: умение пользоваться справками программ и поиском в Интернете; способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Оборудование и материалы: компьютер, презентационное оборудование.

Планирование занятий

Занятие 1: изучение материала урока № 1 «Знакомство со средой IntelliJ».

Занятие 2: выполнение лабораторной работы № 1 «Знакомство со средой IntelliJ».

2. Первые программы на языке Java. Основные операторы

Рекомендуемое количество часов на данную тему — 6 часов.

Планируемые результаты:

предметные: получение навыков работы с переменными, основными операторами, вводом/выводом данных.

метапредметные: способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Оборудование и материалы: компьютер, презентационное оборудование.

Планирование занятий

Занятие 1: изучение материала урока № 2 «Первые программы на языке Java. Основные операторы».

Занятия 2 и 3: выполнение лабораторной работы № 2 «Переменные. Операторы».

Занятия 4–6: выполнение лабораторной работы № 3 «Ввод данных».

3. Классы

Рекомендуемое количество часов на данную тему — 5 часов.

Планируемые результаты:

предметные: освоение базовых понятий объектно-ориентированного программирования в Java, работа с классами и объектами; развитие умения соотносить данные, полученные в результате декомпозиции предметной области, с соответствующими структурами языка Java.

метапредметные: способность анализировать предметную область, выявлять взаимодействия между составными частями простых систем; способность производить простейшую декомпозицию предметной области, способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: эстетическое отношение к языкам программирования, осознание их выразительных возможностей, готовность и способность обучающихся к саморазвитию и личностному самоопределению; готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Оборудование и материалы: компьютер, презентационное оборудование.

Планирование занятий

Занятие 1: изучение материала урока № 3 «Классы и объекты».

Занятия 2 и 3: выполнение лабораторной работы № 5.1 «Классы».

Занятия 4 и 5: выполнение лабораторной работы № 5.2 «Статические элементы».

4. Управляющие структуры

Рекомендуемое количество часов на данную тему — 6 часов.

Планируемые результаты:

предметные: получение навыков по работе с управляющими структурами в Java; умение применять логические операции при построении алгоритмов; умение правильно разработать алгоритм и реализовать его с учётом особенностей языка Java.

метапредметные: способность к самостоятельному поиску информации; способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: эстетическое отношение к языкам программирования, осознание их выразительных возможностей; готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Оборудование и материалы: компьютер, презентационное оборудование.

Планирование занятий

Занятие 1: изучение материала урока № 4 «Управляющие структуры. Последовательные инструкции. Ветвления».

Занятия 2 и 3: выполнение лабораторной работы № 4 «Управляющие структуры. Последовательные инструкции. Ветвления».

Занятие 4: изучение материала урока № 5 «Управляющие структуры. Циклы».

Занятия 5 и 6: выполнение лабораторной работы № 6 «Управляющие структуры. Циклы».

5. Структуры данных

Рекомендуемое количество часов на данную тему — 6 часов.

Планируемые результаты:

предметные: получение навыков работы с массивами и списками в языке Java; развитие навыков применения массивов и списков для хранения и структурирования информации.

метапредметные: способность к самостоятельному поиску информации, в том числе умение пользоваться справками программ и интернет-поиском; способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: эстетическое отношение к языкам программирования, осознание их выразительных возможностей; готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Оборудование и материалы: компьютер, презентационное оборудование.

Планирование занятий

Занятие 1: изучение материала урока № 6 «Структуры данных».

Занятия 2 и 3: выполнение лабораторной работы № 7 «Массивы».

Занятия 4 и 5: выполнение лабораторной работы № 9.1 «Списки».

Занятие 6: выполнение лабораторной работы № 9.2 «Списки».

6. Работа со строками

Рекомендуемое количество часов на данную тему — 2 часа.

Планируемые результаты:

предметные: получение навыков работы со строковыми данными в Java.

метапредметные: способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Оборудование и материалы: компьютер, презентационное оборудование.

Планирование занятий

Занятия 1 и 2: выполнение лабораторной работы № 8 «Работа со строками».

7. Отладка кода

Рекомендуемое количество часов на данную тему — 2 часа.

Планируемые результаты:

предметные: получение навыков работы с отладчиком кода среды IntelliJ.

метапредметные: способность к самостоятельному поиску информации, в том числе умение пользоваться справками программ и интернет-поиском; способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: эстетическое отношение к языкам программирования, осознание их выразительных возможностей; готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Оборудование и материалы: компьютер, презентационное оборудование.

Планирование занятий

Занятие 1: изучение материала урока № 7 «Отладка кода».

Занятие 2: выполнение лабораторной работы № 10 «Отладка кода».

Лабораторные работы

Все лабораторные работы выполняются в среде IntelliJ Community Edition. Для создания Java-проектов необходимы установленная платформа JDK и сама среда IntelliJ. Процесс установки всех компонентов подробно описан в справочных материалах.

Для выполнения работ нет необходимости создавать отдельные папки для проектов на жёстком диске. По умолчанию новые проекты IntelliJ создаются в пользовательской папке текущего пользователя Windows, (например, C:\Users\Shamil\IdeaProjects). Предполагается, что у каждого ученика будет создаваться свой личный аккаунт в ОС Windows, соответственно все папки с проектами будут уникальными для каждого ученика. Есть и другой вариант — создать отдельные папки на жёстком диске для работы с проектами. В этом случае каждый ученик будет сам ответственен за контроль папок проектов, каждую нужно будет создавать в ручном режиме, что не так удобно. Данный вопрос остаётся на усмотрение учителя исходя из его личных предпочтений и необходимости дальнейшего управления проектами учеников различных курсов.

Задания, помеченные звездочкой (*), являются либо заданиями повышенной сложности, либо для их выполнения необходим материал из следующих занятий. В этом случае можно вернуться к выполнению таких заданий позже.

Лабораторная работа № 1. Знакомство со средой программирования IntelliJ. Создание первого проекта

Теоретическая часть

Для создания первого проекта необходима установка JDK и IntelliJ IDEA. Процесс установки описан в справочных материалах.

Под проектом Java в среде IntelliJ понимается папка, содержащая подпапки и файлы с кодом, настройками, ресурсами и т. д. У проектов имеется определённая структура расположения папок и файлов. Для любого проекта существует возможность экспорта на другие носители и соответственно импорта в другие версии среды или на другой компьютер с установленной средой IntelliJ. При импорте проекта в IntelliJ в некоторых случаях необходимо в настройках уточнить или заново указать версию JDK. Обычно появляется всплывающее окно, в котором одним щелчком мыши можно выбрать нужную версию. Также можно зайти в настройки проекта через пункт меню *File — Project Structure*.

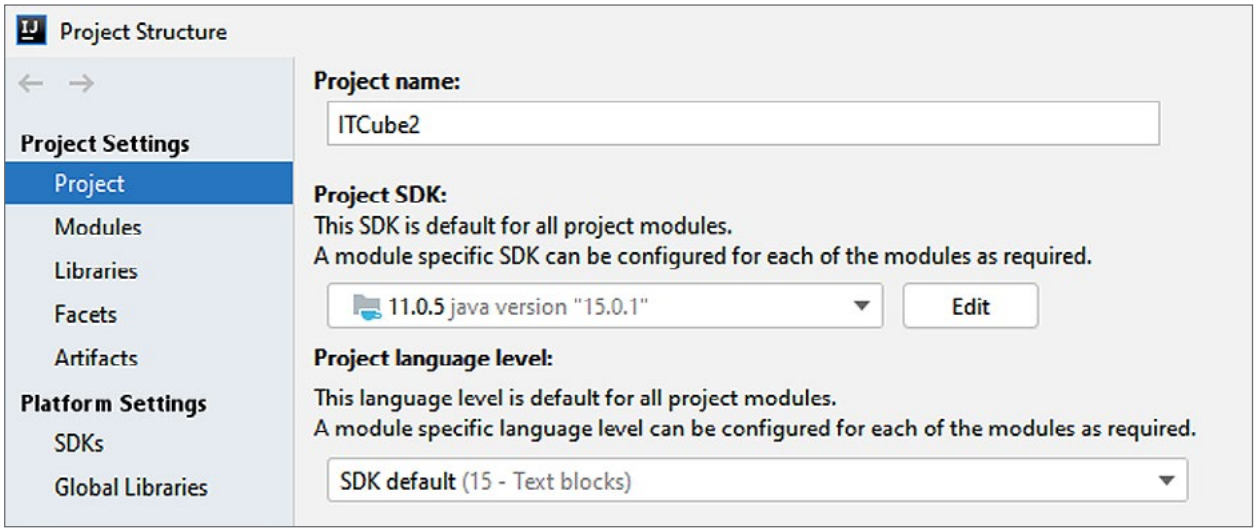

Рис. 17. Окно выбора версии JDK для проекта

В графе *Project SDK* нужно выбрать необходимую версию JDK. Выбор возможен только при наличии хотя бы одной установленной версии JDK. Если выбор отсутствует, то необходимо обратиться к системному администратору лаборатории IT-куб или проверить наличие установленной версии платформы JDK, если вы работаете за своим домашним компьютером.

Традиционно при изучении программирования первой задачей является написание программы, выводящей на экран фразу «Hello, world!». Для реализации такой программы на Java необходимо выполнить следующую последовательность действий.

1. Запустить среду IntelliJ. При первом запуске IntelliJ IDEA появляется стартовое окно.

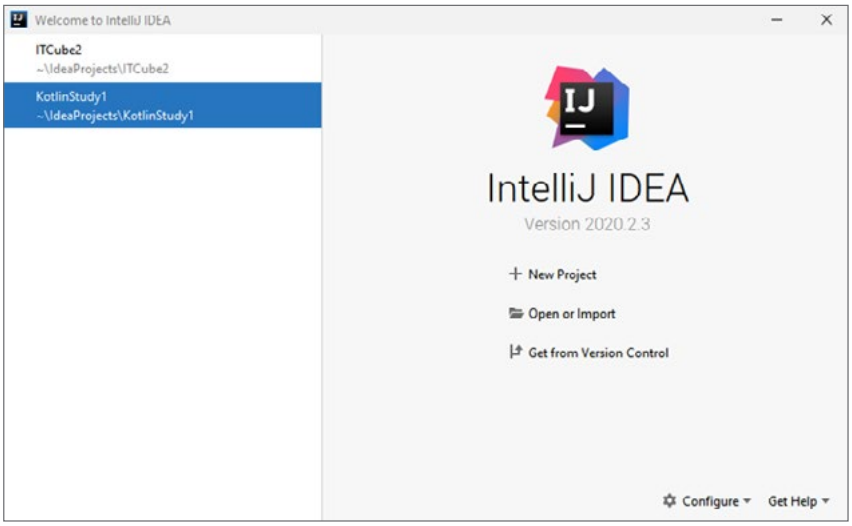

Рис. 18. Стартовое окно среды IntelliJ

2. Создать новый проект IntelliJ.

Для создания нового проекта нужно выбрать пункт *New Project*.

Замечание: от версии к версии интерфейс меню и пунктов выбора в меню у IntelliJ может незначительно отличаться. В целом последовательность действия и логика работы приложения остаётся неизменной.

Если пользователь уже разрабатывал какие-то проекты в среде IntelliJ, то вместо стартового окна может сразу открыться последний проект, с которым он работал. В этом случае для создания нового проекта необходимо воспользоваться меню: *File — New — Project*.

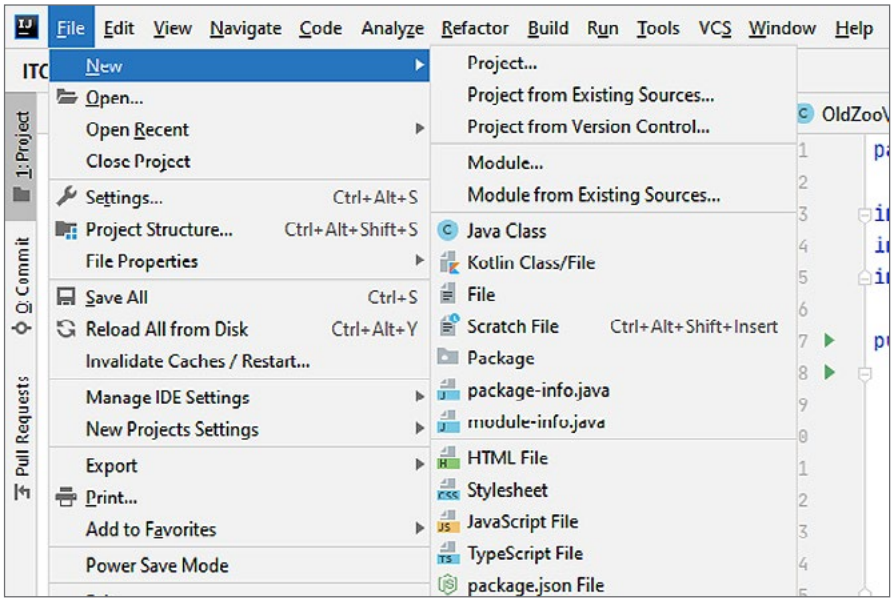

Рис. 19. Создание нового проекта через меню

Далее в появившемся окне нужно слева выбрать *Java*, убедиться, что в графе *Project SDK* указана необходимая версия платформы JDK и справа внизу нажать кнопку *Next*.

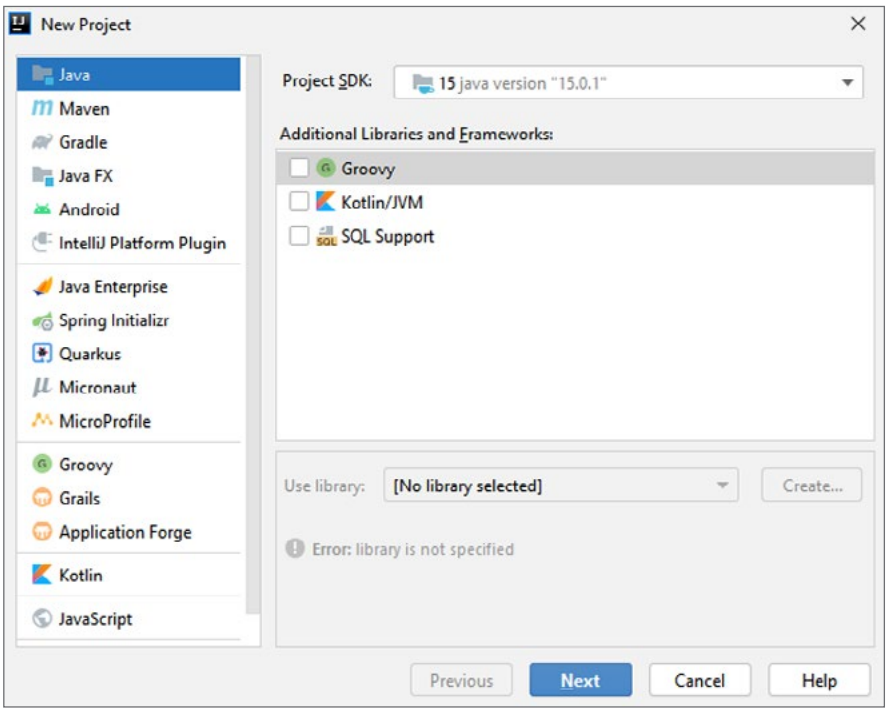

Рис. 20. Окно создания нового проекта IntelliJ

Если JDK не установлен, то возможен вариант упрощённой установки платформы через раскрывающийся список в графе *Project SDK*. Для этого нужно выбрать в списке пункт *Download JDK* и в появившемся диалоговом окне выбрать необходимую версию JDK.

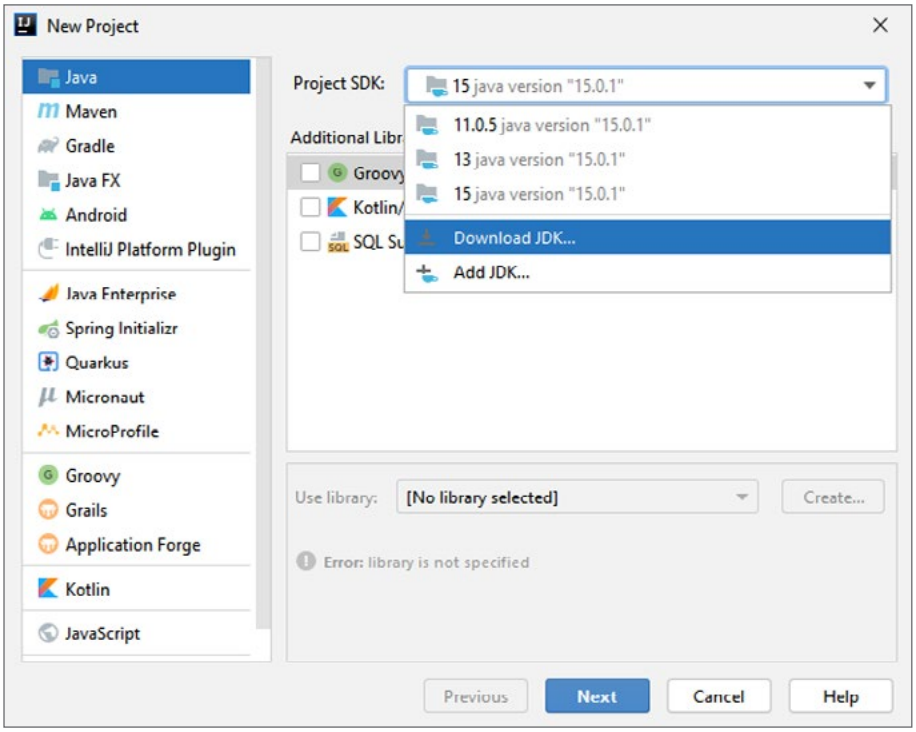

Рис. 21. Автоматическая установка JDK при создании нового проекта в IntelliJ

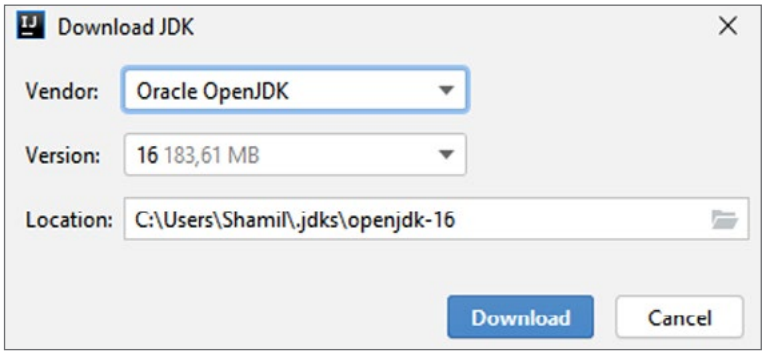

Рис. 22. Меню выбора версии JDK для автоматической установки

В появившемся окне можно выбрать создание проекта из шаблона. Для этого нужно отметить галочкой *Create project from template*.

По умолчанию в IntelliJ есть только один шаблон проекта — «Command Line App», который предполагает запуск приложения как консольного. Для подтверждения нужно нажать кнопку *Next*.

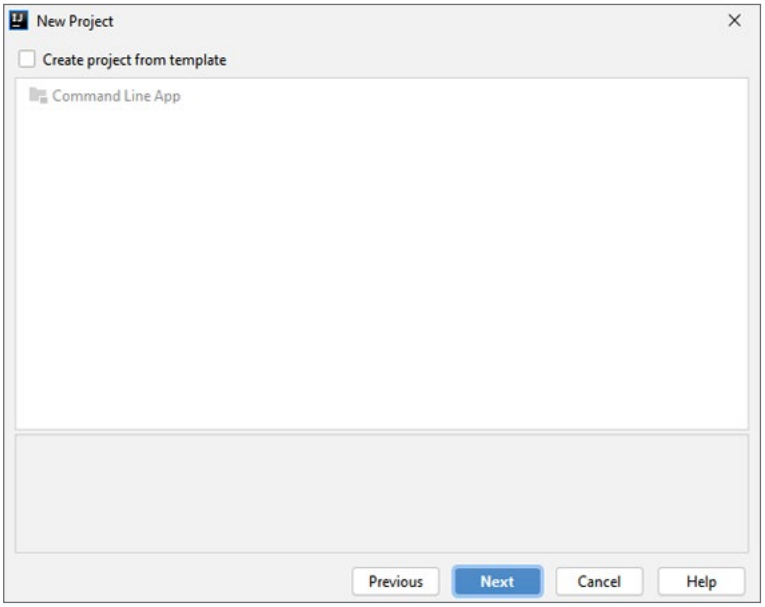

Рис. 23. Окно выбора шаблона проекта

В появившемся окне выбора названия и расположения проекта нужно ввести название для проекта, например «ITCubeFirstProject». Также можно указать структуру папок (пакетов) проекта. По умолчанию это «com.company», его можно поменять, например, на «com.itcube». Также можно указать путь к расположению проекта (по умолчанию путь определяется автоматически и лучше его не редактировать, о чём было написано ранее). Далее нажать кнопку *Finish*.

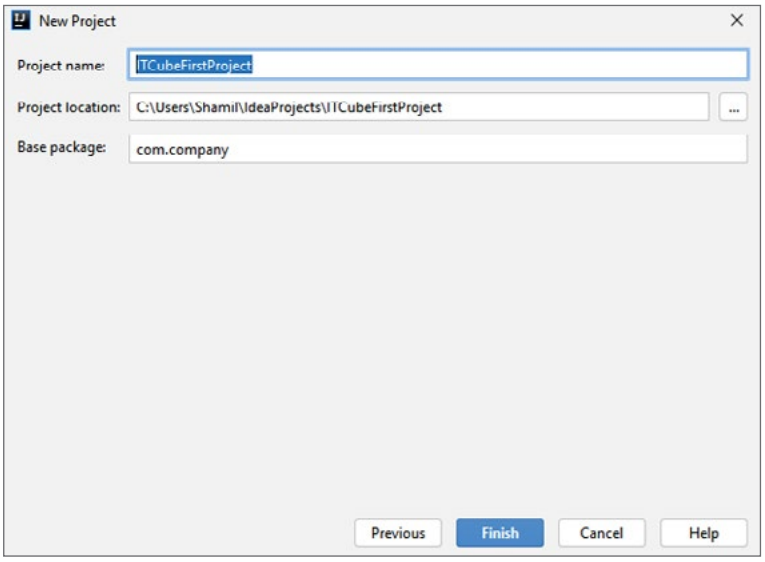

Рис. 24. Окно выбора названия и расположения проекта

3. Записать команду, выводящую на экран сообщение «Hello, world!».

В созданном проекте слева находится структура текущего проекта «ITCubeFirstProject». Все файлы с исходным кодом размещаются в папке «src» (сокращение от слова *sources* — файлы с исходным кодом). Скомпилированный код размещается в папке «out». Папка «src» по умолчанию содержит папку с названием, указанным на предыдущем этапе («com.itcube»), та, в свою очередь, содержит только один файл — Main.java.

При создании проекта изначально файл Main.java содержит только следующий код:

```
public class Main {
     public static void main(String[] args) {
      // write your code here
     }
}
```

По умолчанию в проекте Java присутствует специальный метод `main()`, который является точкой входа в программу и особым образом обрабатывается виртуальном машиной Java. Таким образом, чтобы выполнить какой-либо код, достаточно написать его в методе `main()`. После завершения последней строки кода в методе `main()` выполнение программы завершается.

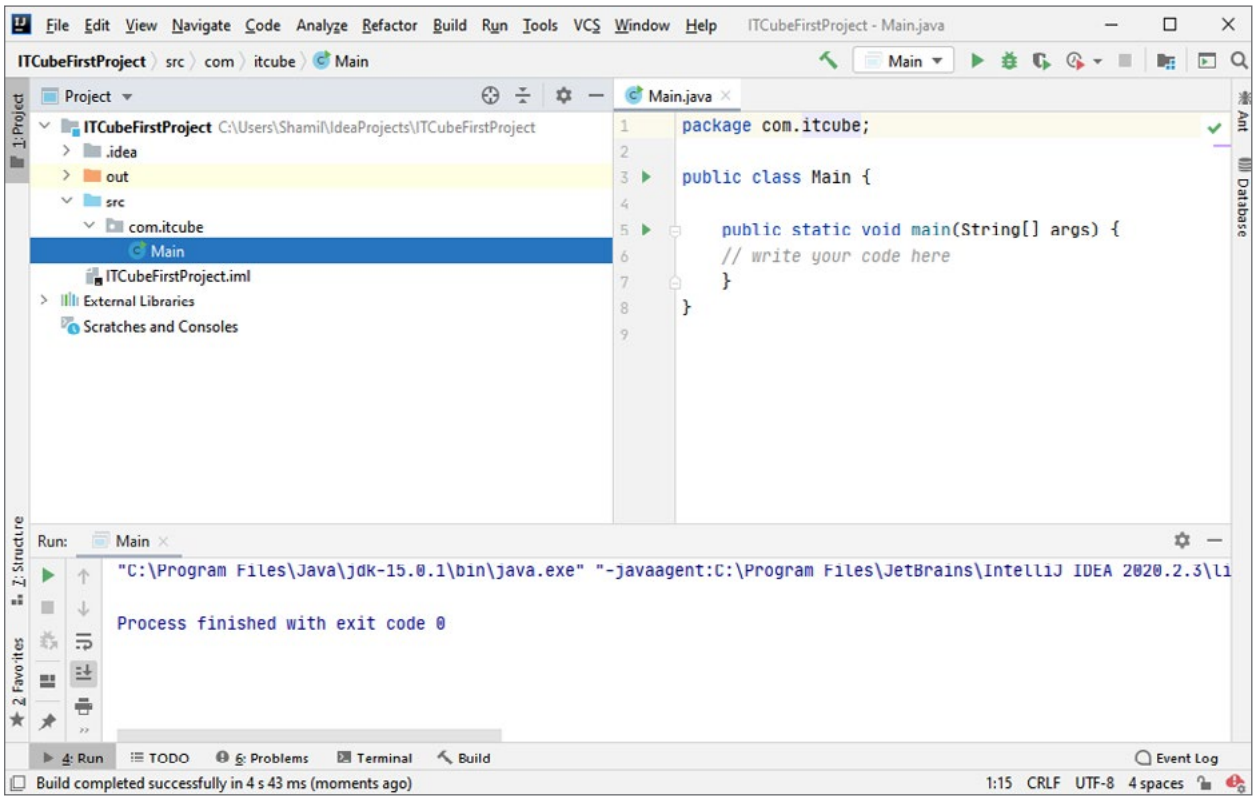

Рис. 25. Окно нового Java-проекта в среде IntelliJ

Для вывода сообщения «Hello, world!» в консоль необходимо вместо комментариев `//write your code here` (на рис. 25 — 6-я строка кода) ввести код:
`System.out.println("Hello, world!");`

4. Запустить программу на выполнение.

Для выполнения кода нужно либо нажать на кнопку с зелёным треугольником  , расположенную в правом верхнем углу окна, либо использовать комбинацию клавиш *Shift + F10*.

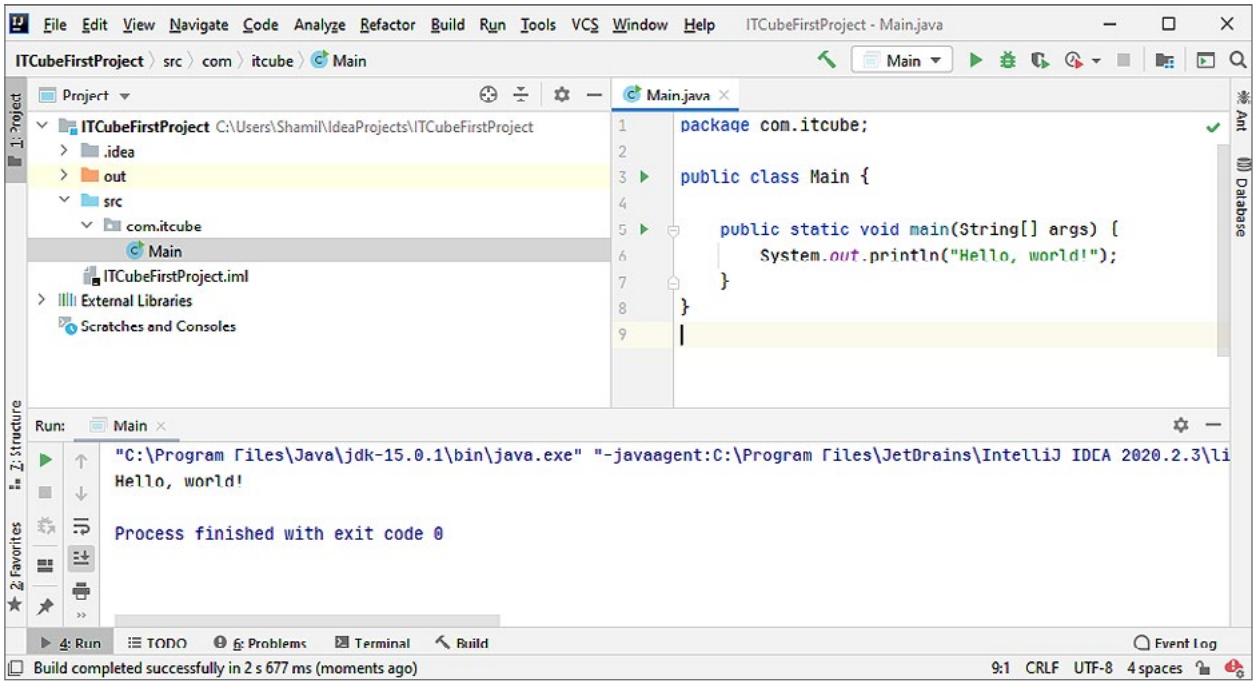

Рис. 26. Результат выполнения кода

Для записи линейного алгоритма достаточно использовать только метод `main()`. Для написания процедурного кода будет необходимо добавлять новые методы и переносить в них фрагменты кода, которые планируется часто использовать. Для написания кода с учётом объектно-ориентированного подхода потребуется добавление в проект новых классов и структур. На данном этапе содержание метода `main()` и класса `Main` можно воспринимать как данность. Достаточно представлять, что метод необходим для запуска программы. Запись `String[] args` нужна для передачи параметров в метод (например, при вызове его через консоль операционной системы); `public` означает, что метод доступен для вызова из любой части проекта; `static` — что для вызова метода `main()` не нужно создавать объект класса; `void` — что метод может только принимать данные и обрабатывать их, но не может возвращать результат.

5. Создать новый класс.
В любом проекте может быть множество классов, файлов, ресурсов.

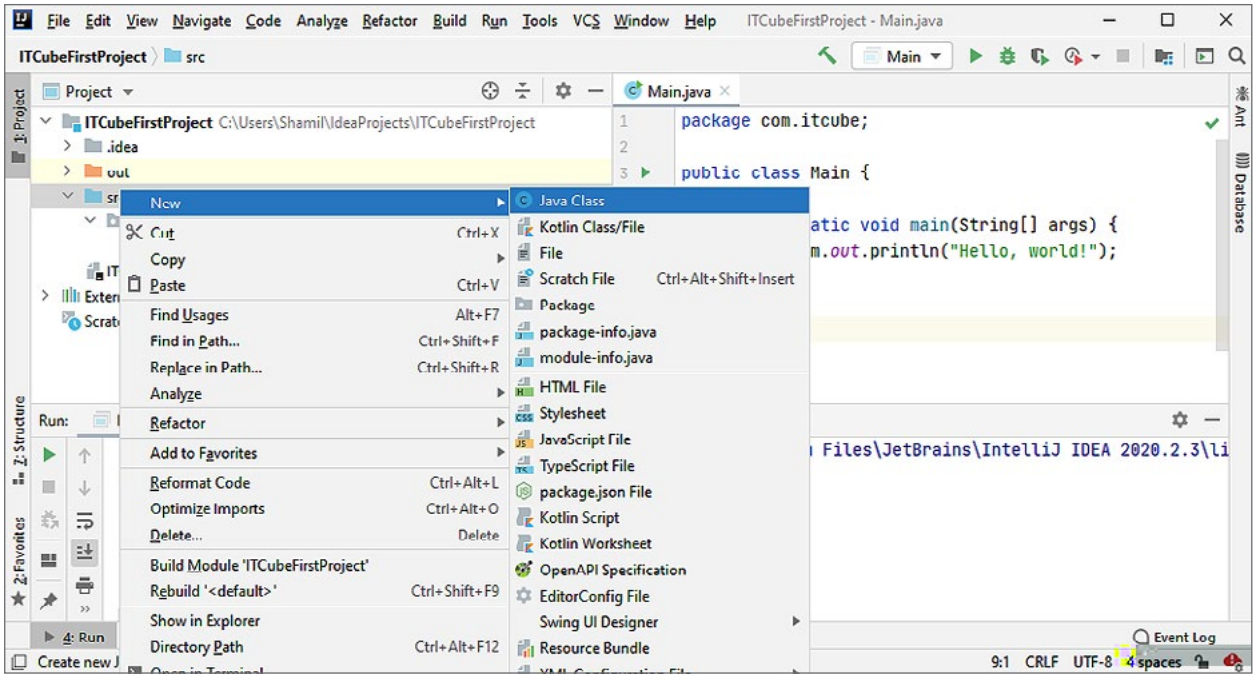

Рис. 27. Создание нового класса в проекте

Чтобы создать новый класс, необходимо правой кнопкой мыши щёлкнуть по папке «src», далее в появившемся списке выбрать пункт *Java Class*, дать название классу (например, «SecondClass») и нажать *Enter* на клавиатуре. Созданный класс также будет храниться в папке «src». В этом классе можно реализовать новый метод `main()` и заполнить его нужным кодом. Для этого необходимо либо вручную прописать такой же метод `main()`, как в классе `Main` выше, либо воспользоваться автозаполнением кода и внутри фигурных скобок класса `SecondClass` ввести символы `psvm` и нажать *Tab* или *Enter*. Результатом будет такой же метод `main()`, как и в классе `Main`, созданный автоматически в шаблоне проекта Java.

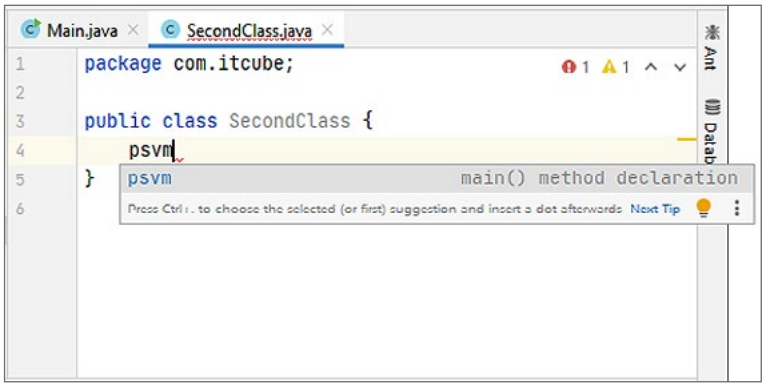

Рис. 28. Использование команды автозаполнения `psvm` для генерации метода `main()`

Рис. 29. Результат автозаполнения в классе `SecondClass`

6. Заполнить новый класс и запустить его код на выполнение.

В новом методе `main()` можно прописать тот же код.

```
System.out.println("Hello, world!");
```

Однако необходимо учесть, что в этом случае при запуске через ▶ или сочетание *Shift+F10* будет вызван тот метод `main()`, который вызывался последним. В данном случае это будет метод `main()` первого класса `Main`.

Чтобы вызвать метод `main()` именно класса `SecondClass`, необходимо в окне структуры проекта щёлкнуть правой кнопкой мыши на этом классе и в контекстном меню выбрать *Run 'SecondClass.main()'*. Результат будет тем же, что и для класса `Main`, поскольку и там, и там написан одинаковый код.

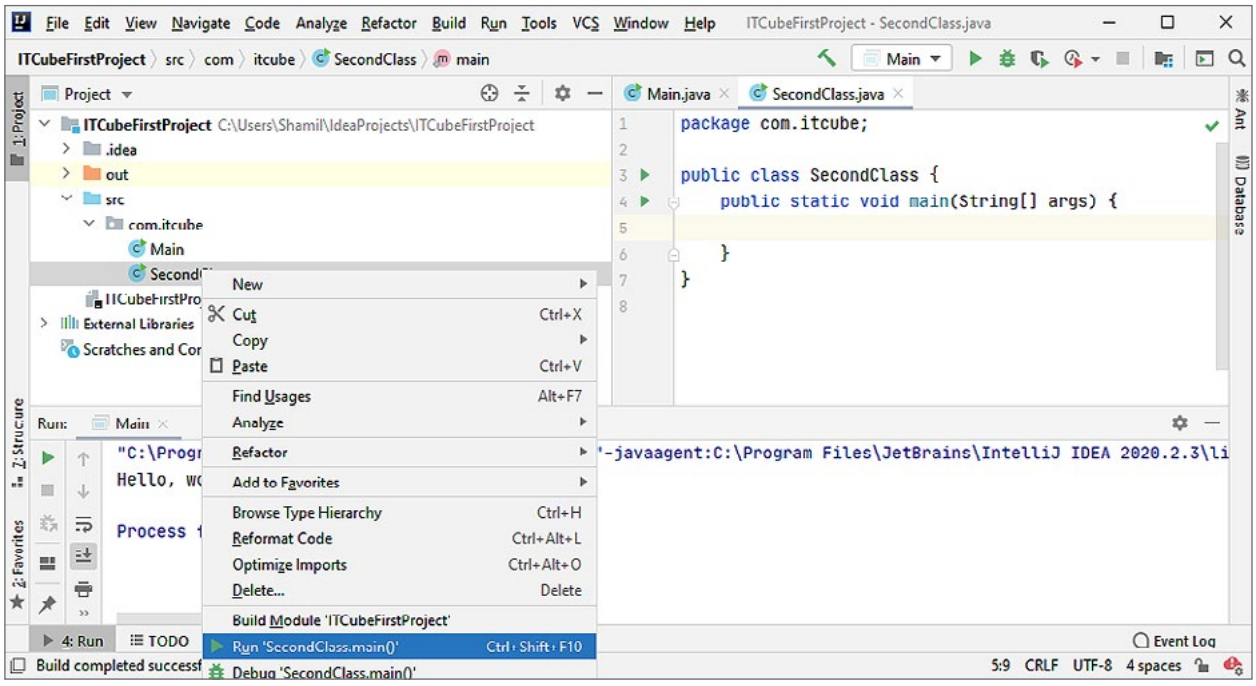

Рис. 30. Вызов метода `main()` у конкретного класса проекта

Средством группировки классов в проектах Java являются также пакеты. Физически пакеты представляют собой обычные папки, а с точки зрения проекта служат пространством имён и средством как физического, так и программно-логического структурирования кода.

Директива `package`, размещаемая в самом начале файла, обозначает принадлежность классов, описываемых в этом файле, указанному пакету. Если класс создаётся в корневом каталоге исходных файлов «src», то директива `package` не требуется. Однако рекомендуется применять эту директиву во всех файлах с целью избежать путаницы в группировании классов.

Практическая часть

Цель работы: ознакомление со средой программирования IntelliJ и создание первого Java-проекта.

Ход лабораторной работы

1. Запустить среду IntelliJ.
2. Реализовать все примеры из теоретической части.
3. Убедиться, что структура пакетов проекта в IDE IntelliJ полностью соответствует структуре папок при просмотре из проводника Windows.
4. Создать программу, которая выводит в консоль текст «Hello, World!».
5. Создать программу, которая выводит в консоль текст «Hello, World!» 3 раза.
6. Удалить у метода `main()` ключевое слово `static` и попробовать запустить программу.
7. Переименовать ранее созданный класс.
8. Удалить ранее созданный класс.
9. Попробовать создать новый проект с другим расположением на жёстком диске.

Выводы: в ходе лабораторной работы было произведено знакомство со средой IntelliJ, произведены базовые действия в среде, создан Java-проект на основе шаблона.

Контрольные вопросы

1. Что такое IntelliJ?
2. Зачем нужен проект?
3. Что вы слышали о компиляторе javac?
4. Для чего нужны интегрированные среды разработки?
5. Какие ещё среды, кроме IntelliJ, вы знаете?
6. Как создать проект?
7. Для чего нужен метод `main()`?

Лабораторная работа № 2. Переменные. Операторы

Теоретическая часть

Для продолжения работы с Java необходимо рассмотреть базовый синтаксис языка. Любая программа на Java так же, как и в других высокоуровневых языках, включает в себя ключевые слова, литералы, переменные, знаки пунктуации, комментарии, правила составления кода. Все требования языка, его синтаксис, ключевые слова, все правила составления кода подробнейшим образом описаны в спецификации Java: https://docs.oracle.com/javase/specs/.

Справочник

Ключевые слова — это специальные зарезервированные слова, посредством которых реализуется программный код. Например: `public`, `static`, `void`, `if`, `else`, `for`, `while`, `break`, `import`, `package`, `new`, `int`, `double` и т. д.

Основные ключевые слова, достаточные для изучения курса

Применение	Ключевые слова
Примитивы	`int, double, boolean, char`
Управляющие структуры	`if, else, for, while, break`
Исключения	`try, catch`
Модификаторы доступа	`public, private, protected`
Классы и интерфейсы	`class, interface, extends, implements, abstract, new, this, super`
Прочее (методы, константы, импорт)	`import, package, void, static, final, return`

Основные правила составления кода следующие.

Комментарии предваряются символами `//` и не влияют на код и выполнения программы. Также можно использовать символы `/**/` для создания длинных многострочных комментариев. В среде IntelliJ комментарии можно вставлять автоматически с помощью сочетания клавиш *Ctrl+Shift+/*.

Блоки исполняемого кода, т. е. инструкции, рассматриваемые вместе, например внутри цикла или условия, внутри класса, метода, блока инициализации, оборачиваются в фигурные скобки `{}`. Фигурные скобки служат обозначением границ, аналогично `begin` и `end` в Pascal, отступам в Python и т. д. Таким образом, фигурные скобки обозначают область действия либо оператора, либо класса или метода в Java, либо отдельного блока кода. Подобный синтаксис типичен для всех C-подобных языков: C, C++, C#, Java и др. Блок кода, содержащийся внутри фигурных скобок, называется телом оператора, метода, класса и т. д. Количество открывающих скобок всегда должно быть равно количеству закрывающих скобок. Для хорошей читабельности кода необходимо придерживаться правил форматирования фигурных скобок. Самый простой вариант выучить эти правила — проанализировать код, получаемый в результате автоформатирования в среде IntelliJ. Для этого достаточно выделить весь код с помощью *Ctrl+A*, затем применить сочетание клавиш *Ctrl+Alt+L*.

Важно!

Все инструкции и выражения заканчиваются точкой с запятой.

```
int x=10;
String s=new String("test");
```

А вот, например, после ключевого слова `for()` или после завершения блока кода цикла `for()` нет необходимости в точке с запятой, в то время, как после каждой инструкции (выражения) в теле `for()` она необходима:

```
for(int i=1; i<10; i++){//здесь знак ";" не нужен
out.println("test");//здесь нужен
} //здесь знак ";" необязателен
```

Таким образом после блоков исполняемых кодов, т. е. после закрывающей фигурной скобки, точка с запятой обычно не нужна, если это не инициализация массива или переменной (т. е. выражение) или объявление анонимного класса. В этом случае внутри фигурных скобок находится не блок исполняемого кода, а специфичная форма синтаксиса.

```
String[] massiv = new String[]{"Hello", "ITCubic"};
```

Справочник

Переменная — это область памяти в компьютере, имеющая имя. Они используются для хранения данных разных типов. Примитивные типы данных предопределены языком и обозначены зарезервированными ключевыми словами.

Основные примитивные типы данных, достаточные для изучения курса

Обозначение	Принимаемое значение	Диапазон	В байтах
`int`	целое число	от -2147483648 до 2147483647	4 байта
`double`	вещественное число	$\pm 4.9 \cdot 10^{-324}$ до $\pm 1.8 \cdot 10^{308}$	8 байт
`boolean`	логическое значение	true/false	4 байта
`char`	символ Unicode	0 до 65535	2 байта

Остальные примитивные типы представляют те же самые виды чисел (целые, вещественные), но с другими диапазонами.

Обозначение	Принимаемое значение	Диапазон	В байтах
`byte`	целое число	от -128 до 127	1 байт
`short`	целое число	-32768 до 32768	2 байта
`long`	целое число	от –9223372036854775808 до 9223372036854775807	8 байт
`float`	вещественное число	$-3.4 \cdot 10^{38}$ до $3.4 \cdot 10^{38}$	4 байта

Смысл использования заданных типов переменных — эффективное распределение переменных в памяти компьютера. Каждая переменная требует место для хранения и ресурсы для обработки.

В рамках данного курса мы будем использовать следующие типы. Для целых чисел — тип `int`, для вещественных (числа с плавающей точкой) — тип `double`, для переменных, принимающих значение истина/ложь (true/false) — тип `boolean`, для символьных переменных — тип `char`.

Пример объявления переменных и присваивания им значений.

```
int number=3;
boolean isChecked=false;
boolean isOk;
double salary;
char mySymbol='c';
isOk=false;
```

Одно из важнейших правил программирования в целом — всегда использовать наиболее осмысленные имена переменных. Нежелательно, за исключением простейших случаев, присваивать переменным названия типа: *x*, *y*, *z*, *a*, *b*, *c*, *m*, *n* и т. д. Это приемлемо только для счётчиков в циклах, которым традиционно присваиваются имена *i*, *j*.

Также существуют ограничения на имена переменных:

- имена переменных не должны совпадать с ключевыми (зарезервированными) словами;
- имена переменных могут начинаться только с букв, нижнего подчеркивания (_) или знака доллара ($);
- остальные символы имени, кроме первого, могут состоять из любых комбинаций букв, цифр, _ и $;
- Java — чувствительный к регистру язык, поэтому `x` и `X` будут двумя разными переменными.

Для проверки допустимости использования символа в имени переменной можно применять следующие методы класса `Character`:

```
public static void main(String[] args) {
    System.out.println(Character.isJavaIdentifierStart('%'));
    System.out.println(Character.isJavaIdentifierPart('a')); }
```

Результат:

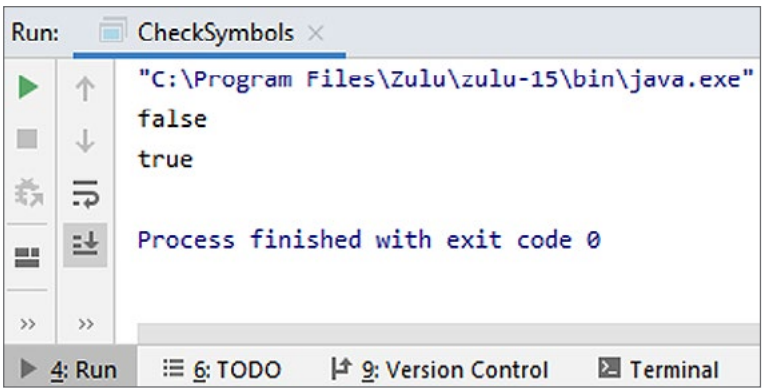

Рис. 31. Результат проверки допустимости символов методами `isJavaIdentifierStart()`, `isJavaIdentifierPart()`

Первый метод проверяет допустимость символа для первой буквы имени переменной, второй — для остальных букв.

Поскольку Java является языком со строгой статической типизацией, то:

- при работе с переменными необходимо учитывать их тип. Например, нельзя к числу прибавить текст, т. е. можно выполнять только те операции, которые доступны для конкретного типа;
- перед использованием переменные должны быть объявлены и иметь конкретный тип. Язык Java не распознаёт автоматически, какого типа может быть переменная, для каких данных она предназначена. Программист сам должен конкретно указать их назначение. За счёт этой особенности языка много ошибок выявляется ещё на этапе компиляции программы, т. е. до того, как программа запустится. Например, в IntelliJ подобные ошибки сразу будут подчёркнуты красным, и среда не даст выполнить код, пока ошибки не будут исправлены.

Особенностью переменных примитивных типов в Java является то, что они передаются по значению, а не по ссылке.

```
int x=3;
int y=x;
x=x+1;
out.println(x); //выведется 4
out.println(y); //выведется 3
```

В этом примере видно, что значение переменной `y` не изменилось после увеличения `x` на единицу, хотя изначально `y` было присвоено значение `x`. Эти переменные не зависят друг от друга и при присваивании передают только свои значения, а не адреса или ссылки на ячейки в памяти, где они находятся.

Другой вид переменных в Java — это переменные объектного типа (или ссылочного типа). Они представляют собой объекты классов (структур, которые будут рассмотрены в следующих темах). В отличие от переменных примитивного типа объектные переменные передаются по ссылке. Это означает, что при присваивании одной переменной другой, они начинают указывать на одну область пространства и одно и то же значение. Более подробно этот механизм будет рассмотрен далее.

Если переменная объявлена с ключевым словом `final`, то такая переменная является константой.

```
public final double PI=3.141;
```

Соответственно, такая переменная должна быть проинициализирована сразу при объявлении и не может быть изменена в дальнейшем.

Также переменные могут быть объявлены с ключевым словом `static`. В этом случае они являются статическими. Подробно данный материал разобран в лабораторной работе № 10 «Статические элементы».

Справочник

Оператор — конструкция языка, определяющая команду (набор команд) языка программирования, задающая выполнение действий. Действие оператора производится над **операндами**, обычно являющимися переменными.

В данном курсе рассматриваются и применяются следующие операторы: арифметические операторы, операторы сравнения, логические операторы, операторы присваивания.

Один из главных операторов, уже использованный в предыдущей лабораторной работе — оператор присваивания `=`. Он используется для присваивания значений переменным любых типов. В Java также существуют сокращённые формы записи арифметических выражений. При их выполнении одновременно с арифметической операцией происходит операция присваивания. Например, выражение `x+=10` является сокращённой формой выражения `x=x+10`.

Сокращенные формы записи арифметических выражений

`+=`	Увеличение левой части оператора на величину правой
`-=`	Уменьшение левой части оператора на величину правой
`*=`	Умножение левой оператора части на величину правой
`/=`	Деление левой части оператора на величину правой

```
import static java.lang.System.out;
public class Operators3 {
    public static void main(String[] args) {
```

```
        int x = 0;
        int y = 10;
        out.println("x="+x+", y="+y+"\n"); //x=0, y=10
        out.println("x+=y,\t x="+(x+=y)); //x=10
        out.println("x+=y,\t x="+(x+=y)); //x=20
        out.println("x+=y,\t x="+(x+=y)); //x=30
        out.println("x-=y,\t x="+(x-=y)); //x=20
        out.println("x*=y,\t x="+(x*=y)); //x=200
        out.println("x/=y,\t x="+(x/=y)); //x=20
    }
}
```

Примечание: сочетания символов `\t` и `\n` используются для форматирования выводимого текста. `\t` означает табуляцию, `\n` применяется для перевода строки.

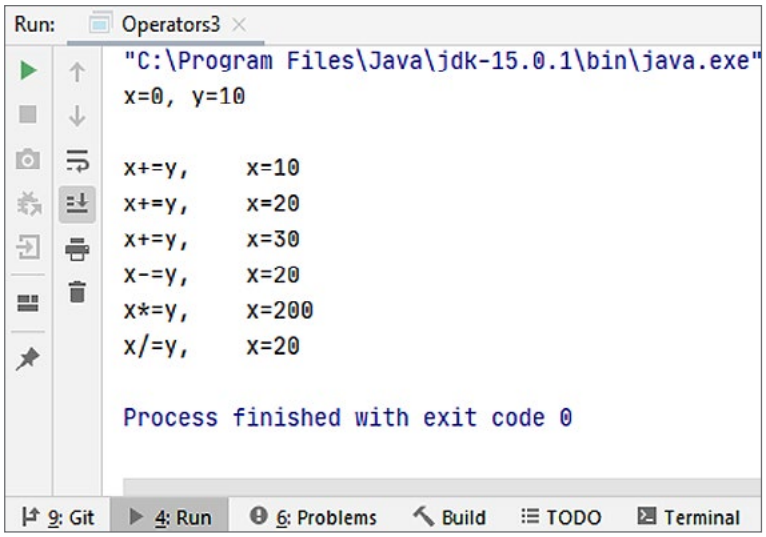

Рис. 32. Результат работы программы

Типовая работа с операторами на языке Java соответствует работе с математическим выражениями из школьного курса математики. Это касается как арифметических операторов, так и группировки при помощи скобок.

Базовые арифметические операторы

Оператор	Описание
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Деление с остатком
++	Инкремент (увеличение на единицу)
--	Декремент (уменьшение на единицу)

С помощью операторов инкремента и декремента создаются упрощённые формы выражений приращения значений переменных, например: выражение `i++` является

упрощённой формой выражения `i=i+1` или `i+=1`, а выражение `i--` является упрощённой формой выражения `i=i-1` или `i-=1`.

Важное свойство инкремента и декремента в Java. У инкремента и декремента существует постфисная и префиксная записи: `x++` и `++x`, `x--` и `--x`. Разница между ними в том, что постфиксная операция выполняется после основного выражения. То есть в случае `y=x++` сначала `y` присваивается значение `x`, и только потом значение `x` увеличивается на единицу. Это важно запомнить, чтобы избежать ошибок в вычислениях. Это же касается и работы инкремента и декремента в операторе вывода. Поэтому в следующей программе используется префиксная запись вида `++x`.

Программа, демонстрирующая результаты арифметических операций Java:

```
import static java.lang.System.*;
public class Operators {
    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        out.println("x="+x+", y="+y+"\n"); //x=10, y=20
        out.println("x+y=\t"+(x+y)); //x+y= 30
        out.println("x-y=\t"+(x-y)); //x-y= -10
        out.println("x*y=\t"+(x*y)); //x*y= 200
        out.println("y/x=\t"+(y/x)); //y/x= 2
        out.println("x%y=\t"+(x%y)); //x%y= 0
        out.println("y%x=\t"+(y%x)); //y%x= 0
        out.println("++x=\t"+(++x)); //++x= 10
        out.println("--x=\t"+(--x)); //--x= 11 } }
```

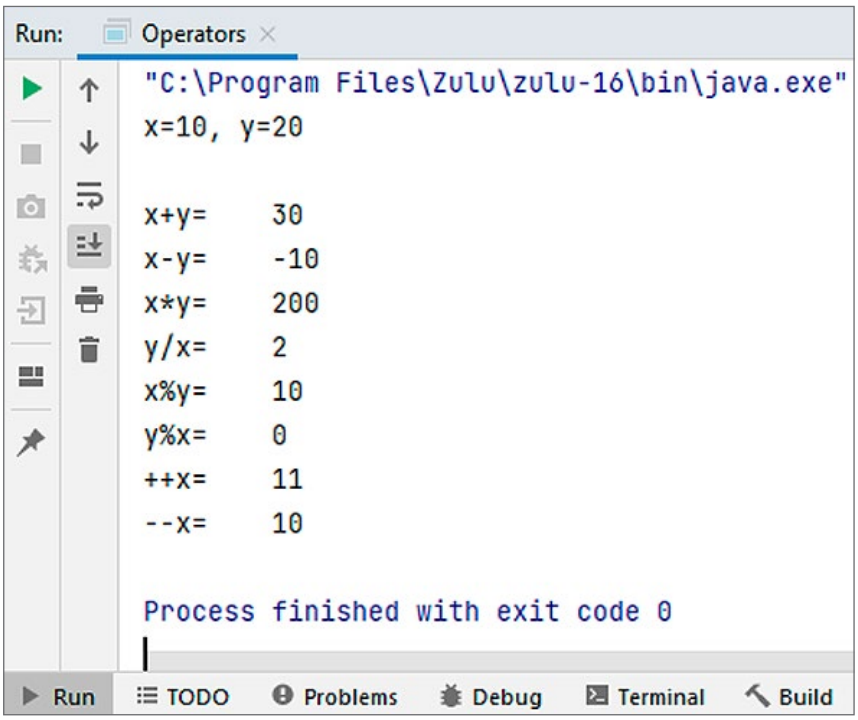

Рис. 33. Результат работы арифметических операторов

Операторы сравнения служат для сравнения значений переменных.

Операторы сравнения

Оператор	Описание
==	тождество
!=	неравенство
>	больше
<	меньше
>=	больше или равно
<=	меньше или равно

```
import static java.lang.System.out;
public class Operators {
    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        out.println("x="+x+", y="+y+"\n"); //x=10, y=20
        out.println("x==y\t"+(x==y)); //x==y false
        out.println("x!=y\t"+(x!=y)); //x!=y true
        out.println("x>y\t\t"+(x>y)); //x>y false
        out.println("x<y\t\t"+(x<y)); //x<y true
        out.println("y>=x\t\t"+(y>=x)); //y>=x true
        out.println("y<=x\t\t"+(y<=x)); //y<=x false } }
```

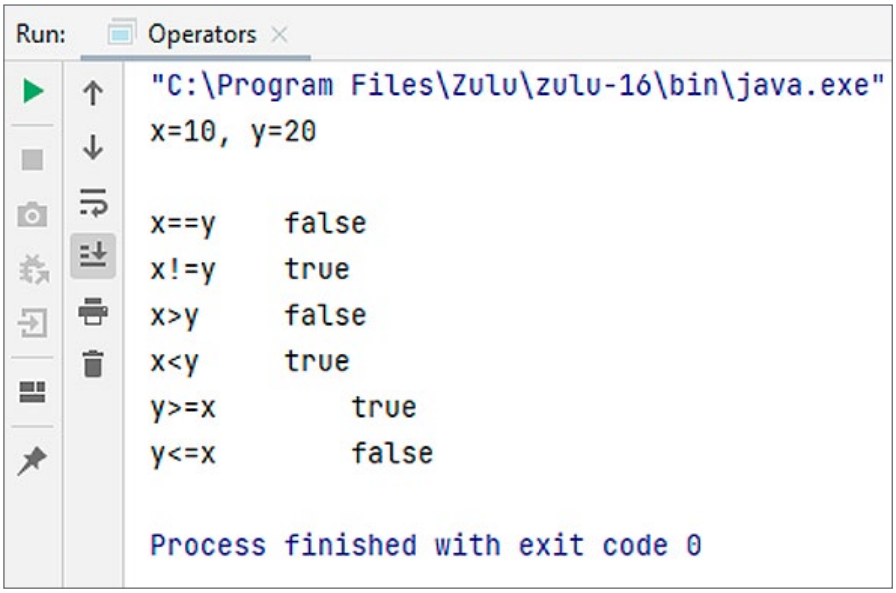

Рис. 34. Результат работы операторов сравнения

Ещё одна группа операторов, используемых для проверки условий, — логические операторы. Они, вместе с оператором Элвиса (`?:`) рассмотрены в лабораторной работе № 4 «Управляющие структуры».

Приоритеты базовых арифметических операторов такие же, как в математике. То же самое касается и скобок, с помощью которых операнды и операторы группируются в вы-

ражении. Скобки повышают приоритет операции. При отсутствии скобок сначала выполняются операции с более высоким приоритетом.

Приоритеты операторов в порядке убывания

()	.	[]	выражения в скобках и точки
!	++	--	отрицание; декремент; инкремент
*	/	%	умножение; деление; остаток от деления
+	-		сложение; вычитание
==	!=		тождественность; неравенство
^			исключающее ИЛИ
&&			логическое И
||			логическое ИЛИ

Для арифметических вычислений в Java будут полезны два вспомогательных класса из пакета стандартной библиотеки.

Класс `Math` содержит математические функции. Чтобы просмотреть список функций и выбрать нужную, в среде IntelliJ достаточно в методе `main()` набрать `Math`.

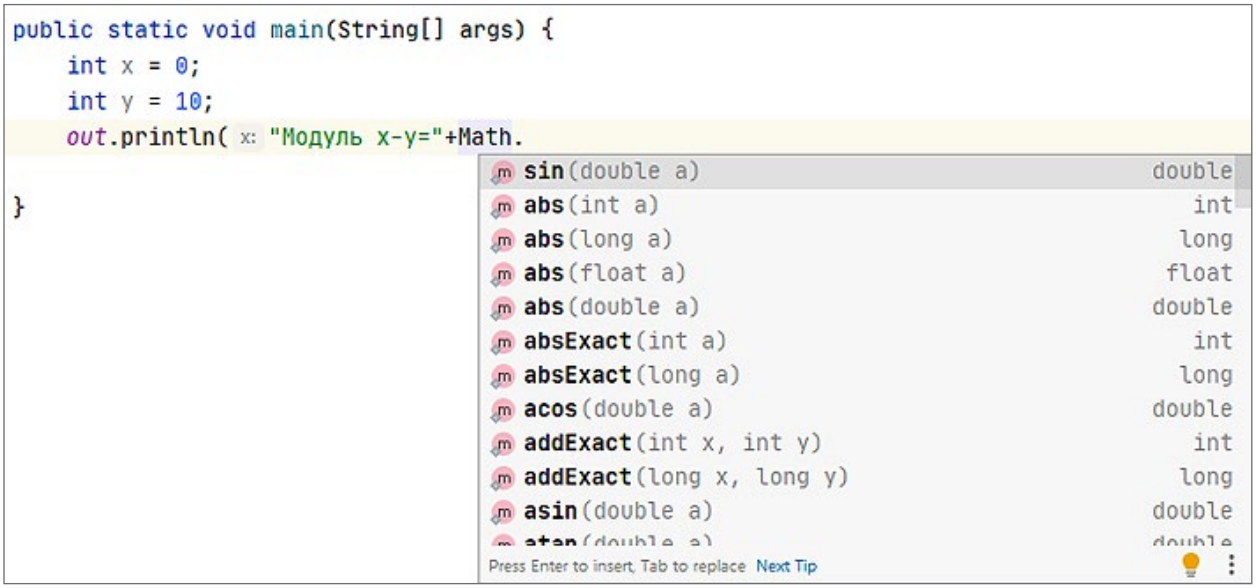

Рис. 35. Выбор функций класса `Math`

```
import static java.lang.System.out;
public class MathAndRandom {
    public static void main(String[] args) {
        int x = 2;
        int y = 10;
        out.println("Модуль (x-y)=\t" + Math.abs(x - y));
        out.println("x в степени y\t"+Math.pow(x,y));
        out.println("sin(90)=\t\t"+Math.sin(Math.toRadians(90)));
        out.println("Число Пи=\t\t"+Math.PI);
    }
}
```

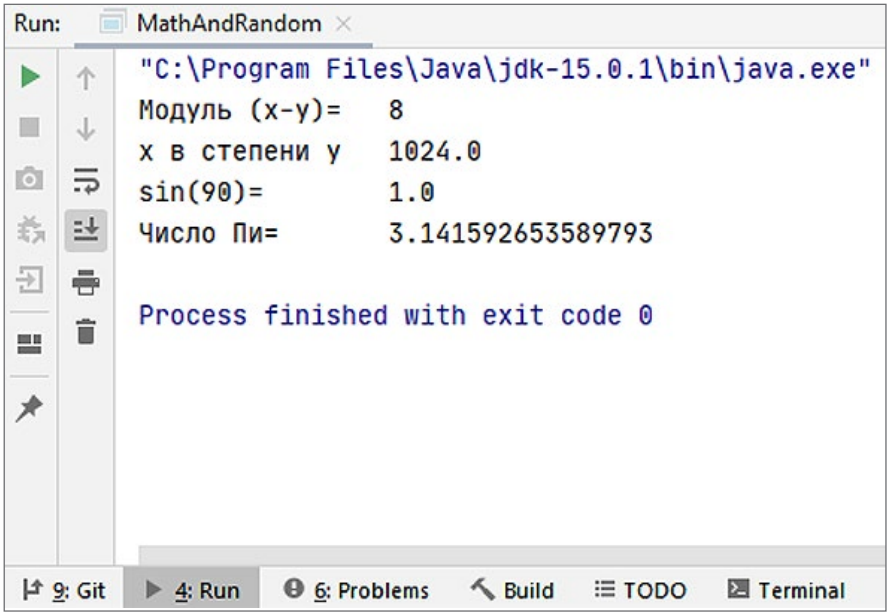

Рис. 36. Результат работы функций класса `Math`

Некоторые функции класса `Math`

`Math.pow(x,y)`	возведение в степень
`Math.sqrt(x)`	корень из числа
`Math.toRadians(grad)`	перевод градусов в радианы
`Math.sin(rad)`	синус (от радиан)
`Math.cos(rad)`	косинус (от радиан)
`Math.tan(rad)`	тангенс (от радиан)
`Math.abs(x)`	модуль числа
`Math.hypot(x,y,…)`	квадратный корень суммы квадратов своих аргументов

Класс `Random` позволяет генерировать псеводслучайные числа. Также можно использовать и метод `Math.random()` для генерирования псевдослучайных чисел от 0 до 1.

```
import static java.lang.System.out;
import java.util.Random;
public class RandomExample {
    public static void main(String[] args) {
        Random random = new Random();
        out.println(random.nextInt(9));
        //случайное целое число от 0 до 9
        out.println(random.nextDouble());
        //случайное вещественное число от 0 до 1
        out.println(random.nextBoolean());
        //случайное значение true/false
        out.println(Math.random());
        //случайное вещественное число от 0 до 1
    }
}
```

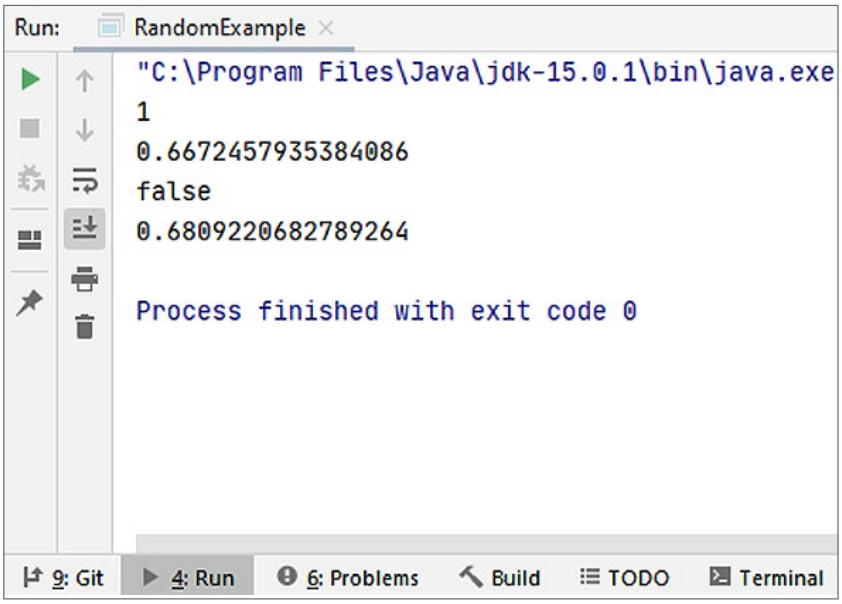

Рис. 37. Получение псевдослучайных чисел

Практическая часть

Цель работы: ознакомление с базовым синтаксисом Java.

Ход лабораторной работы

1. Запустить среду IntelliJ. При необходимости создать новый проект и пустой класс с методом `main()`.

2. Реализовать все примеры, приведённые выше. Проанализировать код.

3. Написать в методе `main()` программу для нахождения периметра и площади треугольника и вывода результата в консоль. Дано: значения катетов *a* и *b* прямоугольника, заданные в виде переменных типа `int`.

4. Написать в методе `main()` программу нахождения корней уравнения и вывода результата в консоль. Дано: параметры *a*, *b* и *c* квадратного уравнения, заданные в виде переменных типа `double`.

5. Написать в методе `main()` программу, реализующую калькулятор с семью операциями: умножение, деление, сложение, вычитание, нахождение синуса, косинуса, нахождение квадратного корня. Ввод чисел организовать с помощью методов класса `Scanner`.

6. Написать в методе `main()` программу для нахождения индекса массы тела по формуле, учитывающей рост и вес, и вывода результата в консоль. Найти формулу самостоятельно в открытых источниках.

7. В методе 7, используя константу `Math.PI`, рассчитать площадь круга. Дано: радиус круга в виде переменной типа `int`. Найти формулу самостоятельно в открытых источниках. Вывести результат в консоль.

Выводы: в ходе выполнения лабораторной работы произведено ознакомление с базовым синтаксисом языка Java.

Контрольные вопросы

1. Что такое оператор?
2. Перечислите основные операторы Java.
3. Что такое ключевое слово?
4. Можно ли переменной дать такое же имя, как у ключевого слова?

5. Какими методами можно проверить, подходят ли конкретные символы для использования в имени переменной?
6. Какие бывают виды операторов?
7. Для чего нужен класс `Math`?

Лабораторная работа № 3. Ввод данных

Теоретическая часть

Данные, необходимые для работы программы, можно задавать как в самой программе, например, присваивая значения объявленным переменным — инициализировать переменную (`int x=3;`), так и в качестве аргументов или ключей при вызове программы.

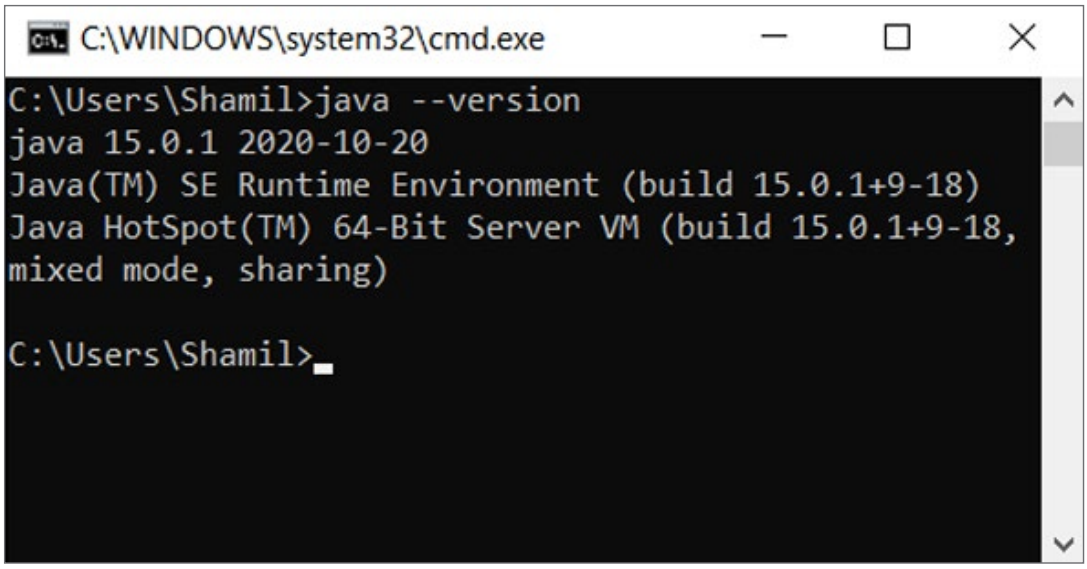

Рис. 38. Пример вызова приложения java.exe с ключом --version

Другой вариант задавать данные — интерактивное взаимодействие с пользователем программы. В этом случае пользователь может вводить данные через консоль, задавать значения переменных и влиять на поток управления программы. В языке Java есть несколько встроенных инструментов, которые позволяют достаточно эффективно работать с пользовательским вводом данных через консоль.

Один из них — это класс `Scanner` из библиотеки java.util стандартного пакета библиотек Java. Он удобен для чтения и обработки вводимых через консоль значений базовых типов.

Основные методы считывания данных класса `Scanner`

`next()`	считать слово
`nextLine()`	считать всю строку
`nextInt()`	считать целое число
`nextDouble()`	считать вещественное число
`nextBoolean()`	считать логическую переменную

Каждый вызов метода типа `next...` приводит к ожиданию ввода значения в консоль и нажатия клавиши *Enter*. Пока не будет нажата клавиши *Enter*, не произойдёт заверше-

ния считывания данных и перехода на следующую строку кода. Следующий метод типа `next...` не выполнится, пока не завершится выполнение предыдущего.

Каждый из этих методов обеспечивает ровно одно считывание данных из консоли. Если же необходимо обеспечить считывание данных в цикле, до тех пор, пока цикл не прервётся, можно использовать метод `hasNext()` вместе с циклом `while()`. Например, цикл `while(hasNext())` будет бесконечно переходить в режим ожидания ввода, а `while(!hasNext("Q"))` завершится при нажатии на клавишу *Q*.

Объект класса `Scanner`, считывающий из стандартного потока ввода (System.in), создаётся следующей командой.

```
Scanner sc=new Scanner(System.in);
```

Считывание данных из консоли можно осуществить, например, следующим образом:

```
String line=sc.nextLine;
int x=sc.nextInt();
```

Для того чтобы считывать значения конкретного символа из строки, можно использовать следующий код:

```
char simvol = sc.next().charAt(/*позиция символа в строке*/);
```

Общие принципы создания проектов и работы в среде IntelliJ изложены в лабораторной работе № 1 «Знакомство со средой IntelliJ».

Практическая часть

Цель работы: ознакомление с консольным способом ввода данных в Java.

Ход лабораторной работы

1. Запустить среду IntelliJ. При необходимости создать новый проект и класс `TestScanner` с методом `main()`. Проанализировать и переписать код программы, предлагающей пользователю ввести свои личные данные (имя и возраст). После ввода данных в консоль должно выводится приветственное сообщение вида: «Привет, Имя. Вам X лет».

```
import java.util.Scanner;
import static java.lang.System.*;

public class TestScanner {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        out.println("Введите ваше имя и возраст:");
        String name = sc.next();
        int age = sc.nextInt();
        out.println("Привет, "+name+". Вам "+age+" лет.");
    }
}
```

Выполнить код, запустив метод `main()` класса `TestScanner`.

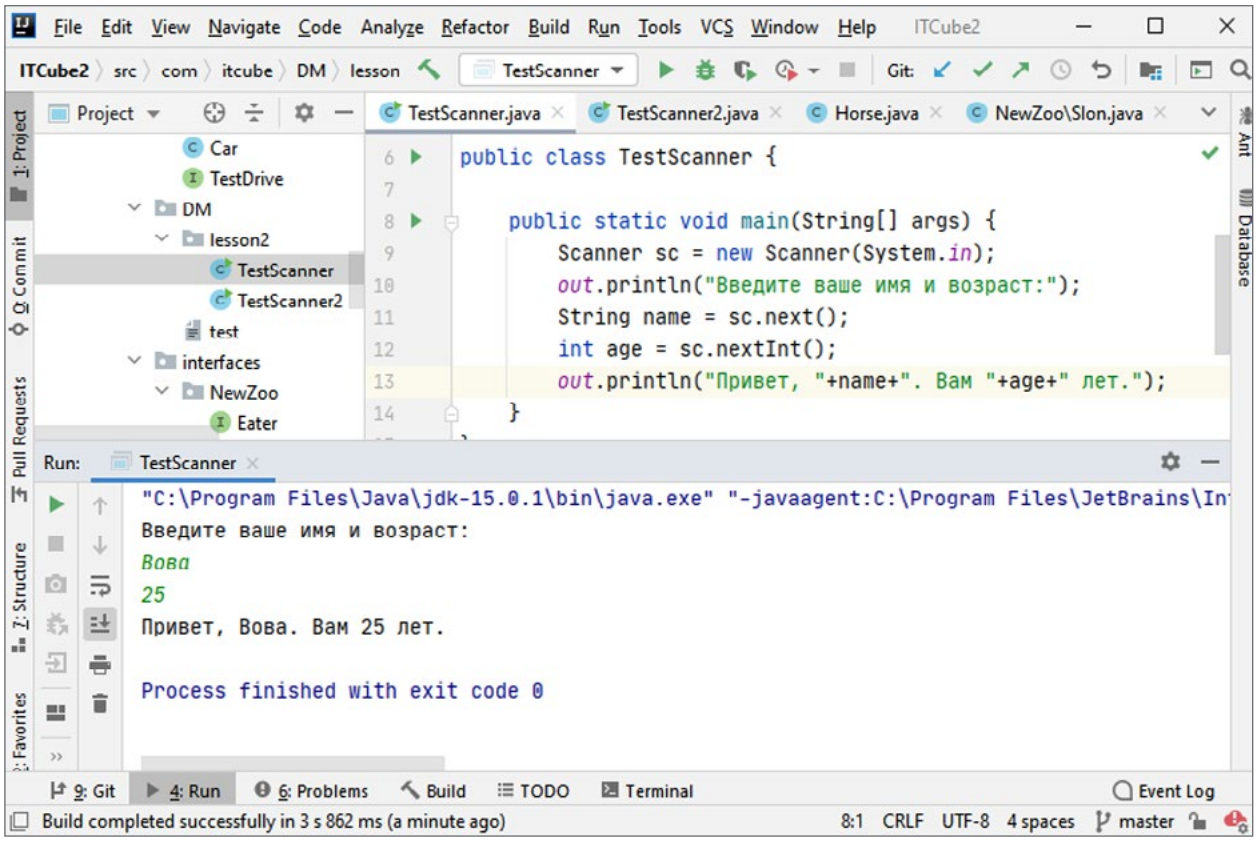

Рис. 39. Результат выполнения программы

Здесь, в отличие от лабораторной работы «Знакомство со средой IntelliJ», в общей структуре кода класса появились дополнительные изменения.

```
import java.util.Scanner;
import static java.lang.System.*;
```

Первая директива `import` необходима для подключения класса `Scanner` в классе `TestScanner` и позволяет использовать конструкцию

```
Scanner sc = new Scanner(System.in);
```

вместо громоздкой конструкции вида

```
java.util.Scanner sc = new java.util.Scanner(System.in);
```

Вторая директива import позволяет использовать конструкцию

```
out.println("текст");
```

вместо более массивной конструкции вида

```
System.out.println("текст");
```

2. Сделать код более интерактивным в плане взаимодействия с пользователем можно, чередуя ввод и вывод. Создать новый класс, например `TestScanner2`. Проанализировать и ввести следующий код. Выполнить и проверить результат.

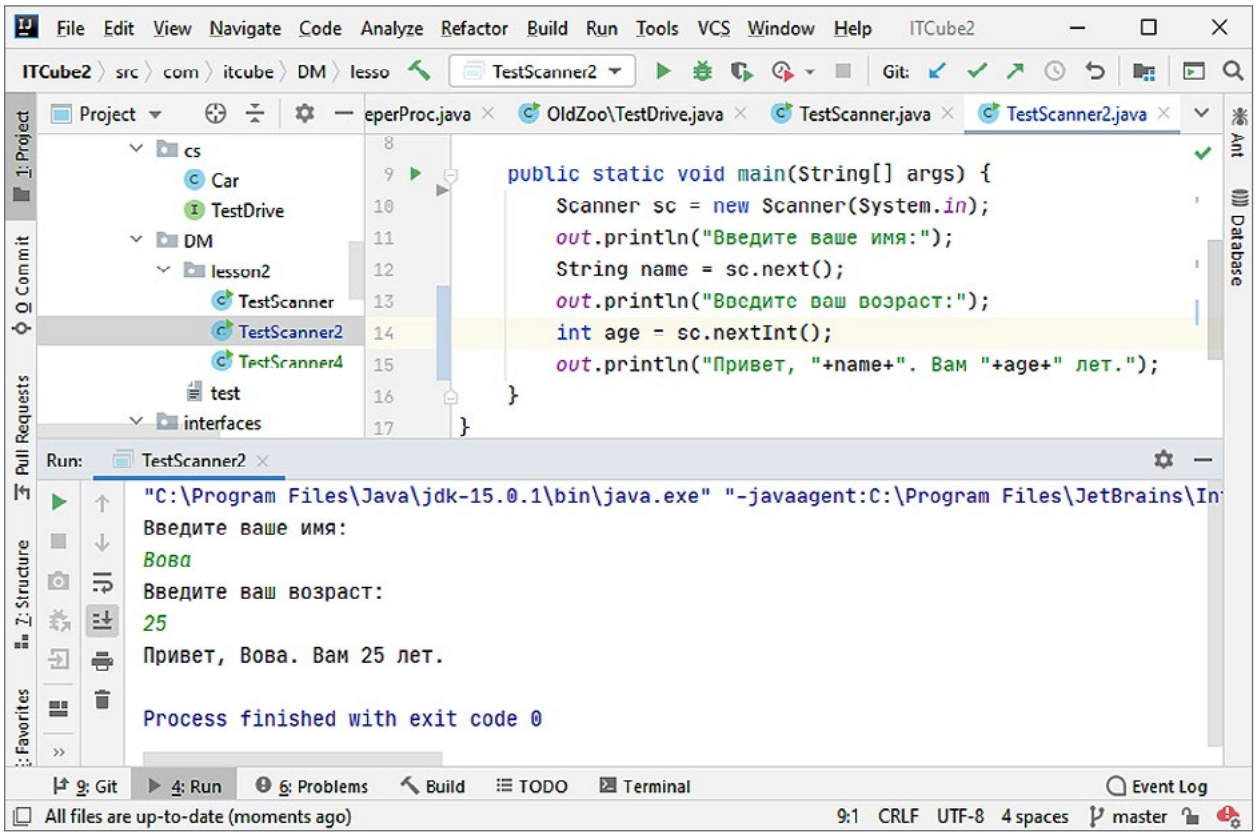

Рис. 40. Чередование ввода и вывода

В случае, когда используется одиночная команда типа `sc.next*`, ожидание ввода прекращается сразу после ввода данных и нажатия на клавишу *Enter*, как и в предыдущих примерах. Часто же бывает необходимо вводить данные в цикле или бесконечно, пока пользователь не нажмет какую-либо клавишу. В этом случае можно применять методы `hasNext()` (для строковых значений), `hasNextInt()` (для целых чисел) и т. д.

3. Создать игру «Угадай число». Имеется заранее указанное целое число. Пользователь путём ввода своих вариантов пытается его угадать. При нажатии на клавишу *Q* происходит выход из программы.

Примерный код может быть следующим:

```
public static void main(String[] args) {
     int guessNumber = 100;
     int tries = 0;
     Scanner sc = new Scanner(System.in);
     out.println("Введите ваше имя:");
     String name = sc.next();
     out.println(name + ", введите ваше целое число:");
     while (sc.hasNextInt()) {
          int number = sc.nextInt();
          tries++;
          if (number != guessNumber) {
```

```
                out.println("Не угадали ... попытка " + tries + "
                        \nВведите ваше число или нажмите любую
                         клавишу для выхода");
            } else {
                out.println("Угадали! Пока!");
                break;
            }
        }
    }
```

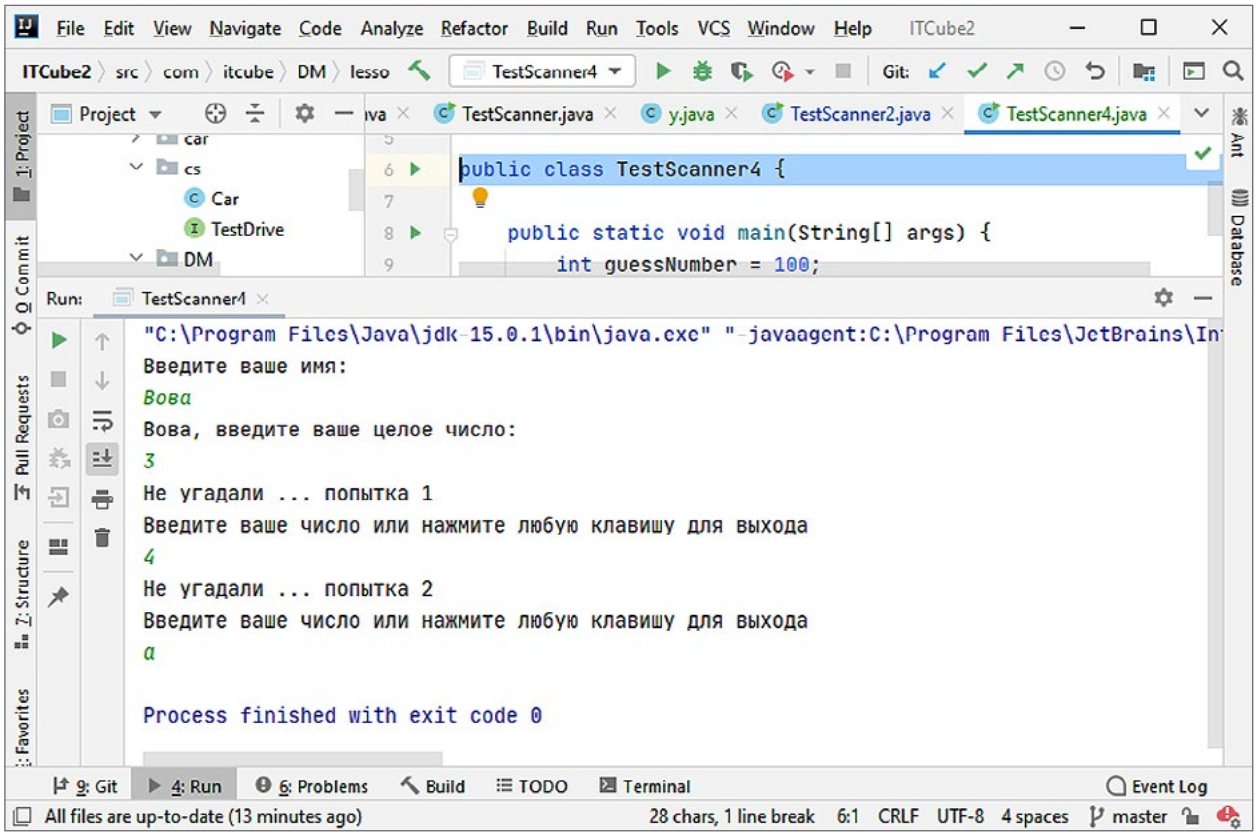

Рис. 41. Результат выполнения кода игры «Угадай число»

4. Усовершенствовать процесс загадывания числа. Вместо присваивания переменной `guessNumber` заданного значения 100 генерировать случайное число от 1 до 20 при помощи класса `Random`:

```
int guessNumber = new Random().nextInt(20);
```

Выполнить код и проанализировать результат.

5.* Сравнить нижеследующий код с кодом, написанным в пункте 3. Запустить. Обсудить отличия.

```
public static void main(String[] args) {
     int guessNumber = new Random().nextInt(20);
     out.println(guessNumber);
     int tries = 0;
     Scanner sc = new Scanner(System.in);
     out.println("Введите ваше имя:");
     String name = sc.next();
     out.println(name + ", введите ваше целое число:");
     String stroka;
     while (!sc.hasNext("Q")) {
          try {
               if (Integer.parseInt(sc.next()) != guessNumber) {
               tries++;
                    out.println("Не угадали ... попытка " +
                                tries + "\nВведите ваше число
                                или нажмите " + "Q для выхода");
               } else {
                    out.println("Угадали!Пока!");
                    break;
               }
          } catch (Exception e) {
          }
     }
}
```

6.* Изменить код в классе `TestScanner4` так, чтобы при вводе пользователем какого-либо значения повторно в консоль выводилось соответствующее предупреждение. Использовать динамический список `ArrayList`.

7.* Изменить код задания 6 так, чтобы при вводе пользователем повторно одного и того же значения более 5 раз программа выводила сообщение «Пользователь не ведает, что творит» и досрочно завершалась.

Выводы: в ходе лабораторной работы были освоены приёмы работы с классом `Scanner` и вводом данных через консоль.

Контрольные вопросы и задания

1. Для чего нужен функционал класса `Scanner`?
2. Как считать целую строку? Как считать символ? Как считать логическую переменную?
3. Самостоятельно изучите информацию о возможности ввода данных посредством класса `BufferedReader`.
4. Какой класс используется в Java для работы с консолью?
5. Чем `println()` отличается от `print()`?
6. Можно ли выводить какие-либо данные в консоль?
7. В какой библиотеке находится класс `Scanner`?
8. Какие данные можно получать из консоли?

Лабораторная работа № 4. Управляющие структуры. Последовательные инструкции. Ветвления

Теоретическая часть

Тема управляющих структур Java рассматривается в трёх лабораторных работах. В данной работе изучаются программирование линейных алгоритмов и разветвляющихся алгоритмов, в следующей — циклы, и далее — вложенные циклы и вложенные управляющие структуры на примере работы с одномерными и двумерными массивами.

Кроме изучения темы управляющих структур, в данной работе необходимо закрепить понимание того, что область действия каждого оператора обозначается фигурными скобками:

`{` — начало области действия,

`}` — конец области действия.

Важно!

Как следует из теоремы Бёма—Якопини, любой алгоритм может быть представлен в виде комбинации трёх структур управления: последовательные инструкции, ветвления и циклы.

Линейные алгоритмы представляют собой последовательные инструкции, выполняющиеся по порядку одна за другой. Таким образом, в реализации таких алгоритмов последовательные инструкции выполняются строка за строкой сверху вниз.

В Java необходимо использовать точку с запятой для разделения последовательных инструкций (см. лабораторную работу № 2).

```
public static void main(String[] args) {
     int x,y;
     String s="test";
     x=5; y=x*2; out.println(x+" "+y); x=10;
     Man m=new Man("Вася");
     m.setAge(95);
     out.println(s);
}
```

Последовательное выполнение также означает, что нельзя выполнить присваивание `x=5`, прежде чем переменная `x` будет объявлена, или же нельзя переменную `y` выражать через переменную `x`, прежде чем `x` будет инициализирована.

Для записи ветвлений используется условный оператор `if()`. Условие записывается в круглых скобках, а инструкция, выполняемая при истинности условия, записывается сразу после оператора:

```
if (a>0)
     b=10;
```

Если в условном операторе имеется только одна инструкция, то фигурные скобки для тела `if()` можно опустить, так как нет необходимости в установлении границ. В этом случае первая после оператора `if()` инструкция автоматически попадает в его область видимости.

Если же необходимо добавить вторую и более инструкции, выполняемые при истинности условия, то в этом случае необходимо их всех заключить в фигурные скобки.

```
if (a>0){
    b=10;
    c=3;
}
```

Если появляется несколько условий, влияющих на выполняемые действия, то может быть использован условный оператор `if-else`. Смысл `if-else` в разветвлении условий, т. е. при выполнении условия выполняется первая инструкция, а если условие не выполняется, то следующая:

```
if (a==8){
    b=2;
}else{
    b=3;
}
```

Существует расширенная версия оператор `if-else`, когда ветвление не раздваивается, а делится на три и более ветви:

```
if (a>10)
    b=10;
else if (a<0)
    b=5;
else if (a==0)
    b=0;
else{
    out.println("Тест");
}
```

Фигурные скобки можно опустить, если в блоке исполняемого кода, относящемся к `else` или `else if()`, содержится только одна инструкция.

Отличие между `else` и `else if()` в том, что `else` подразумевает собой «во всех прочих случаях», а `else if()` подразумевает конкретное условие, которое прописывается в скобках.

В круглых скобках операторов `if()` и `else if()` записывается условие, которое представляет собой логическое выражение либо переменную типа `boolean` и принимает значения true/false.

```
boolean isOk=true;
if (isOk) {
    out.println("isOK=true");
}
if (a>b) {
    out.println("a>b");
}
```

Таким образом, с помощью логических операций можно создавать составные условия. Основные операции — логическое И (`&&`), логическое ИЛИ (`||`) и отрицание (`!`). Логическое И означает, что должны одновременно выполняться все условия.

```
if (a>0 && a<10){
    b=10;
}
```

Данное условие можно трактовать так: если переменная $a > 0$ и при этом $a < 10$, тогда переменной *b* присвоить значение 10.

В случае возникновения необходимости проверить выполнение хотя бы одного из нескольких условий, т. е. достаточно, чтобы выполнилась или первая часть условия, или вторая, или обе части, используется логическая операция ИЛИ (||):

```
if (a>2||a<-10){
    b=0;
}
```

Разумеется, в этом случае условие также может состоять не только из двух частей, а из трёх и более.

```
if (a>2||a<-10||c>10){
    b=0;
}
```

Данное условие можно трактовать так: если переменная $a > 2$, или $a < -10$, или $c > 10$, или ($a > 2$ и $a < -10$), или ($a > 2$ и $c > 10$), или ($a < -10$ и $c > 10$), или ($a > 2$ и $c > 10$), или ($a >$ и $a < -10$ и $c > 10$), тогда переменной *b* установить значение 0.

Если требуется проверить невыполнение условия, то используется логическая операция отрицания (`!`).

```
boolean isOk=true;
if (!isOk) {
    out.println("");
}
```

Исключающее ИЛИ используется реже. Например, если необходимо, чтобы выполнялось только одно условие и при этом не выполнялось второе.

Помимо прочих, существует тернарный оператор, иначе — оператор Элвиса, который представляет собой «синтаксический сахар», позволяющий использовать сокращённую форму записи условия `if-else`. Например, код

```
boolean useSugar=true;
String sugar=useSugar ? "ok" : "not ok";
```

является полным эквивалентом кода

```
boolean useSugar=true;
String sugar;
if (useSugar) {
    sugar = "ok";
} else {
    sugar = "not ok";
}
```

В условный оператор можно вкладывать не только последовательные инструкции, но и другие условные операторы.

```
boolean useSugar=true;
if (useSugar) {
     if (a>b) {
          out.println("sugar and a>b");
     } else
          out.println("sugar and a<b");
} else {
     out.println("Test");
}
```

Очевидно, что могут иметь место не только одинарные вложения условий, но и двойные, тройные и т. д.

Для составления логических выражений удобно использовать таблицу логических операций, с помощью которой можно определить результат действия логических операторов при известных значениях операндов.

Таблица логических операций

При помощи таблицы логических операций можно определить результат действия логических операторов при известных значениях операндов. Например, для логического И (`X&&Y`), если *X*=1 (истина) и *Y*=1 (истина), то результат равен 1 (истина); для логического тождества (`X==Y`), если *X*=1 (истина) и *Y*=0 (ложь), то результат равен 0 (ложь).

X	Y	!X	X&&Y	X||Y	X^Y	X==Y
0	0	1	0	0	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	0
1	1	0	1	1	0	1

Практическая часть

Цель работы: ознакомиться с ветвлениями и логическими операциями в Java.

Ход лабораторной работы

1. Запустить среду IntelliJ. При необходимости создать новый проект и класс с методом `main()`. Проработать все примеры из теоретической части.

2. Написать программу, которая по трём числам *a*, *b*, *c* определяет, могут ли они быть сторонами треугольника, и если да, то вид треугольника. Вывести результат в консоль.

3. Написать программу для нахождения корней квадратного уравнения $ax^2 + bx + c = 0$. На вход программы при помощи класса `Scanner` подаются коэффициенты *a*, *b*, *c*.

4. Написать в методе `main()` программу, выводящую в консоль рекомендации для человека в зависимости от его индекса массы тела. Использовать конструкцию вида `if-else`.

5. Написать в методе `main()` программу, выводящую в консоль рекомендацию о поощрении ученика в зависимости от количества набранных им баллов:

от 0 до 50 — отчислить;
от 51 до 75 — поздравить;
от 76 до 90 — вручить грамоту;
от 91 до 100 — подарить кубик Рубика.

6. Перенести код расчёта индекса массы тела и код выдачи рекомендации из метода `main()` в отдельные методы. Далее в методе `main()` реализовать вызовы этих методов.

Выводы: в ходе выполнения лабораторной работы получено базовое представление о конструкции ветвления в языке Java.

Контрольные вопросы

1. Что обозначают фигурные скобки после операторов `if()`?
2. Если внутри оператора `if()` находится другой оператор `if()`, входит ли внутренний `if()` в область видимости внешнего?
3. За счёт чего можно организовать ветвление в программе?
4. Как можно сократить запись следующего кода из двух операторов `if()`?

```
if(условие1){
     if(условие2){
          действие;
     }
}
```

Лабораторная работа № 5.1. Классы

Теоретическая часть

Тема ООП в Java чрезвычайно обширна. В данном курсе она рассматривается кратко. К тому же с каждой новой версией JDK могут появляться дополнения в механизмах ООП в Java. Полная документация по механизмам ООП в языке Java находится по ссылке https://docs.oracle.com/javase/tutorial/java/concepts/index.html. В данном же пособии рассматривается база, достаточная для реализации типовых задач и анализа кода в рамках курса.

Справочник

Класс — это базовая структурная единица языка Java. Класс является шаблоном для создания объектов, в нём задаются начальные значения переменных и поведение функций и методов.

Пустой класс в Java представляет собой обычный текстовый файл с расширением *.java с подобным наполнением:

```
package com.itcube;
public class MyITCubeFirstClass {
}
```

Фигурные скобки обозначают область действия класса.

Класс содержит свойства (поля) класса и методы класса. Например, класс `Man` (человек) может содержать свойство `height` (рост) и метод `run()` (бежать). Таким образом, свойства обозначают определённые характеристики класса, а методы — определённые действия.

```
  public class Man {
       //поля класса
       public double height, weight; //рост, вес
       public int age; //возраст
       public boolean running=false; //логическая переменная,
показывает, бежит человек или нет
       public double salary; //зарплата
       public String name; //имя

       //методы класса
       public void run() { //устанавливает переменной running
значение "истина"
            running = true;
       }
        public String say() { //возвращает строку с приветствием
return "Hello!";
       }
      public void print(String s) { //выводит строку s в консоль
System.out.println(s);
       }
  }
```

Областью действия полей класса является весь класс. Это значит, что они могут быть использованы в любом месте класса: в любом методе, в любом блоке кода.

Поля класса являются переменными, соответственно определяются так же, как было рассмотрено ранее, т. е. они могут быть примитивами, объектного типа, быть константами, статическими, и т. д.

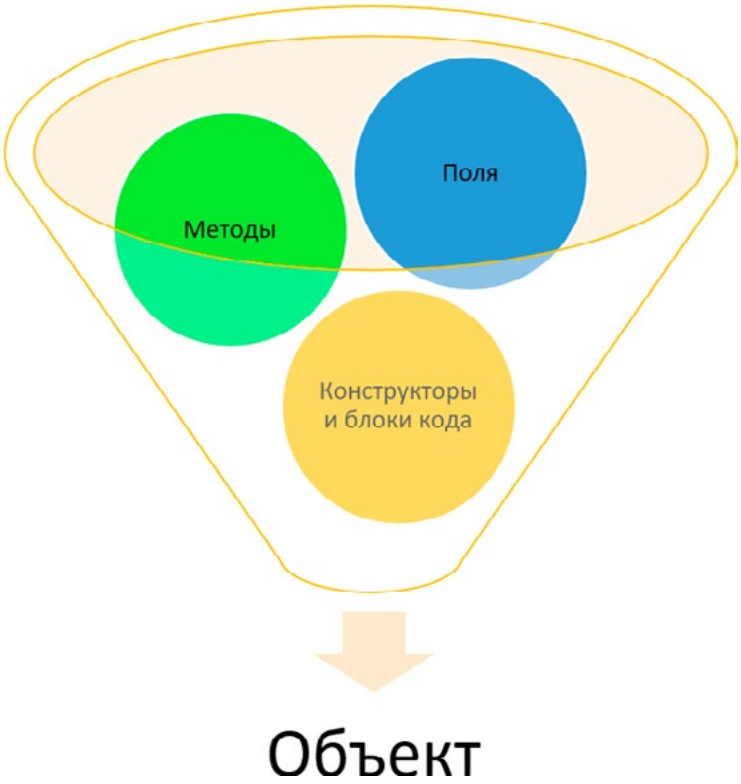

Рис. 42. Схематическое изображение класса

Класс можно представить как чертёж дома, который составляет архитектор. На чертеже указаны параметры дома — площадь, высота, прочие характеристики. А объект, или экземпляр класса — это дом, построенный по этому чертежу. Если быть точнее, то правильнее представить, что класс является чертежом «умного дома», оборудованного электроникой и способного на заранее заданные действия — вывести оповещение, открыть ворота, включить отопление, проветрить помещение и др., так как в классе определяются не только свойства, но и поведение, которое прописывается в методах, конструкторах, блоках инициализации и пр.

Важно!

Применительно к объекту класса также используется слово «тип». Типом объекта является сам класс, на основе которого он создан. В остальных случаях под типом может подразумеваться абстрактный класс или интерфейс.

Чтобы использовать класс и применять его функционал (методы), необходимо создать экземпляр класса, иначе — объект класса. Созданный объект можно использовать: вызывать методы, модифицировать свойства, передавать ссылку на него в другие методы и другие объекты. Чтобы создать объект, необходимо его объявить.

```
Man misha;
```

Здесь `misha` — имя переменной типа (класса) `Man`. Но простого объявления недостаточно, это лишь заявление о том, что переменная с именем misha должна ссылаться на объекты типа класса `Man`, т. е. переменная объявлена, но самого объекта класса пока не существует и под него не выделена память. `misha` — это имя, но это лишь объявление имени, самого объекта ещё не существует. Для создания самого объекта используется ключевое слово `new`:

```
misha = new Man();
```

Или можно объединить обе операции в одну:

```
Man misha = new Man();
```

Без создания объекта класса значение переменной `misha` является `null`, т. е. объекта не существует. При попытке работать с такой переменной Java выбросит исключение `NullPointerException`.

После применения директивы `new` объект создан, он уже существует в динамической области памяти JVM. Процесс загрузки классов и создания экземпляра класса довольно сложный и не отражён в данном пособии. Для продвинутых учеников можно рассказать про области памяти JVM — кучу и стек. При создании экземпляра класса он хранится в куче (динамической области памяти для хранения объектов и их переменных), доступ к переменным и ссылкам на объекты происходит посредством стека. Подробно этот материал можно изучить в статье по ссылке https://topjava.ru/blog/stack-and-heap-in-java.

При инициализации объекта в памяти компьютера создаётся его уникальный идентификатор — хэш-код, который выглядит примерно так: com.itcube.TestClass@4f5e60a1.

После создания объекта можно вызывать его методы и получать или устанавливать значения свойств. Для вызова методов и получения значений свойств используется символ точки, которая разделяет имя переменной объектного типа и название его метода или свойства.

```
import static java.lang.System.*;
public class TestDrive {
    public static void main(String[] args) {
        Man misha = new Man(); //создать новый объект
        misha.name = "Миша"; //установить поле name=Миша
        misha.run(); //вызвать метод run()
        String s = misha.say(); //сохранить в s результат
метода say()
        misha.print(s); //вызывать метод print() с параметром s
        misha.age=30; //установить поле age=30
        misha.salary=30000; //установить поле salary=30000
        out.println(misha.name+ ": age=" +misha.age);
//вывести значение поля age в консоль
        out.println(misha.name+ ": salary="+ misha.salary);
//вывести значение поля salary в консоль
    }
}
```

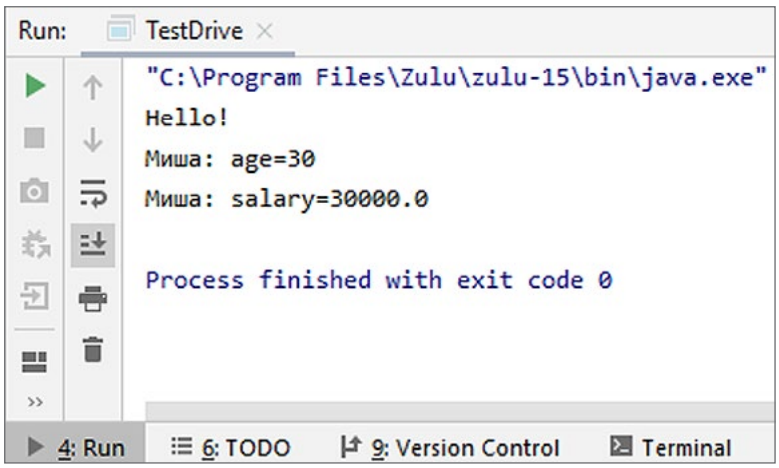

Рис. 43. Результат работы с методами и свойствами класса

Среди операций точка, скобки и квадратные скобки обладают наивысшим приоритетом.

Важно!

В Java всё, кроме примитивов, является ссылками.

Примитивы занимают меньше места в памяти и обрабатываются иначе, нежели объекты. Примитивы передаются по значению, а объекты — по ссылке. Соответственно ссылка лишь указывает на объект, но не является объектом. Ссылка является указателем на объект, поэтому на один объект может указывать и две, и более ссылок. Например, при объявлении `Man misha2=misha` переменная `misha2` указывает на тот же объект в памяти, что и переменная `misha`. Обе ссылки будут абсолютно эквивалентны в плане работы с объектом класса `Man`. Если поменять значение какого-то свойства объекта через переменную `misha`, то при считывании значения этого свойства через переменную `misha2` будет считано обновлённое значение.

```
   public static void main(String[] args) {
        Man misha= new Man(); //создать новый объект
        Man misha2 = misha;
        out.println("age misha=" + misha.age);
        out.println("age misha2=" + misha2.age);
        misha.age=30; //установить значения поля age=30
         out.println("age misha2="+misha2.age); //misha2.age также
стало 30
   }
```

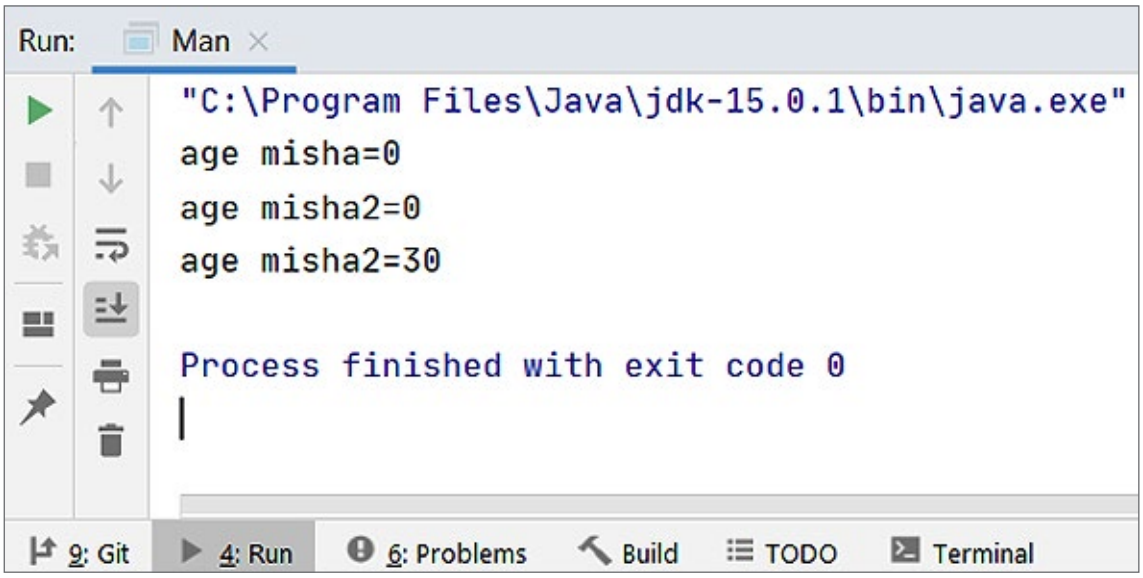

Рис. 44. Результат работы со свойствами объекта через несколько переменных, указывающих на этот объект

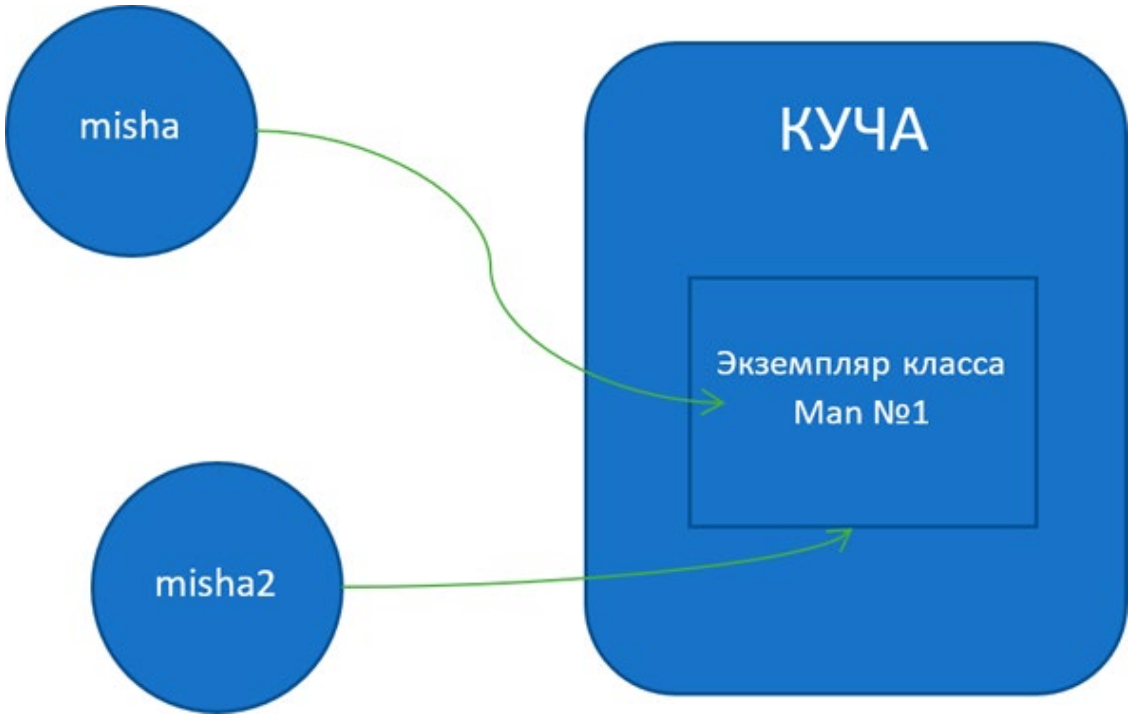

Рис. 45. Переменные объектного типа, ссылающиеся на один объект

В случае же использования переменных примитивного типа, как указывалось в предыдущих лабораторных работах, присваивание значения одной переменной ведёт лишь к изменению её значения, т. е. переменные не начинают ссылаться на один и тот же адрес в памяти. И в дальнейшем при изменении одной переменной значение второй меняться не будет.

```
public class Main {
    public static void main(String[] args) {
        int x, y;
        x=1;
        y=100;
        System.out.println("x="+x+"; y="+y);
        x=y; //переменные примитивного типа: присваивание
        y=200;
        System.out.println("x="+x+"; y="+y);
    //x независим от y, не меняет значение на 200
        Man misha=new Man();
        Man misha2=new Man();
        System.out.println(misha.toString() +"
"+misha2.toString());
       misha=misha2; //переменные объектного типа: присваивание
        System.out.println(misha.toString() +"
"+misha2.toString());
    //хэш-коды misha и misha2 равны, значит, это один и тот же
объект
     }
}
```

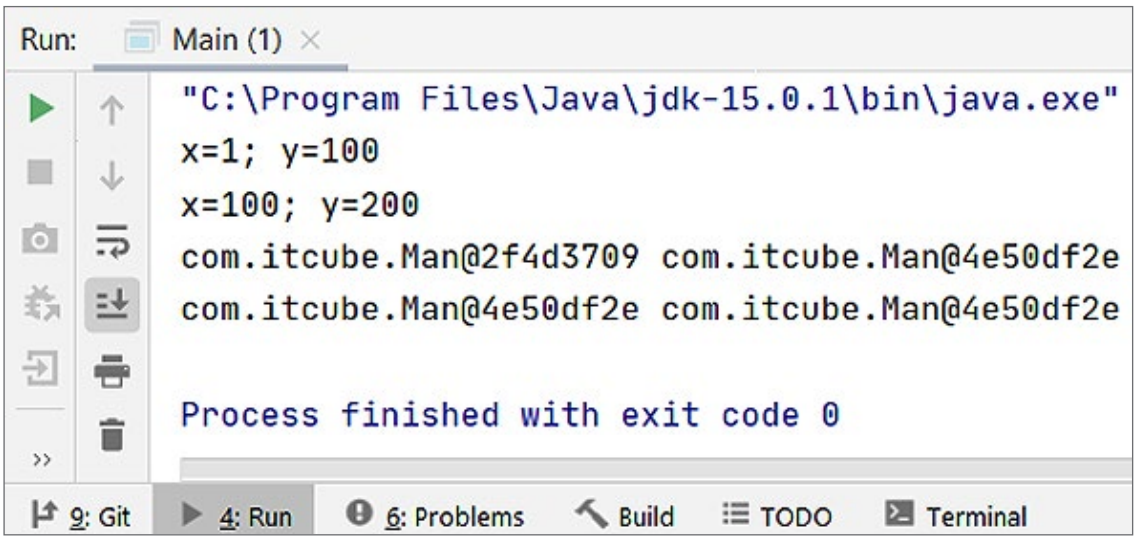

Рис. 46. Хеш-коды объектов равны

Как видно, хеш-коды (уникальные идентификаторы) объектов, на которые указывают `misha` и `misha2`, стали равны, что означает, что они указывают на один и тот же объект.

Для классов фигурные скобки имеют тот же смысл, что и для операторов, т. е. обозначают границы области действия класса. Всё, что находится внутри фигурных скобок класса, относится к этому классу.

Справочник

Методы состоят из операторов, имеют набор входных аргументов. У каждого непустого метода есть цель, например: отправить файл по почте, рассчитать параметр по формуле, нарисовать рисунок и передать его пользователю в виде объекта, вызвать по шаблону ряд других методов и т. д. Функционал метода определяется содержащимися в теле метода инструкциями.

При выполнении метода выполняется последовательность действий в теле метода: объявление переменных, работа управляющих структур, создание объектов, вызовы других методов, подключение библиотек, расчёт формул, параметров и т. д. Если метод пустой, т. е. его тело не содержит инструкций, то при его вызове ничего не будет происходить.

Пример описания методов класса.

```
package com.itcube.animal;
public class Wolf {
   public String name; //имя волка
     private double weight; //вес волка private int age;
//возраст волка
     private boolean running; //логическая переменная,
обозначающая, бежит волк или нет
     public final int speed=10; //константа, задающая скорость
бега
     public void run() {
          running=true;
          System.out.println(name+" убежал.");
     }
     public String sound() {
          return "Звук волка";
     }
     public String hello() {
          String s=name+" поздоровался";
          return s;
     }
     private int getAge() {
          return age;
     }
     public void setName(String name) {
          this.name=name; }
     public double calcDistance(double hours) {
          double distance=0;
          if (running) {
               distance=speed*hours;
          }
          return distance;
     }
}
```

Фигурные скобки метода обозначают область действия метода. Переменные, объявленные в методе, имеют область видимости, равную области действия метода.

В классе `Wolf` существуют следующие методы:

`run()`	устанавливает значение логической переменной running в true и выводит текст «имя волка убежал» в консоль
`sound()`	возвращает строковое значение «Звук волка»
`hello()`	возвращает строковое значение с именем волка
`getAge()`	возвращает значение возраста волка
`setName()`	устанавливает новое значение имени волка
`calcDistance()`	рассчитывает дистанцию, исходя из количества часов, потраченных волком на путь

Сигнатурой метода называется имя метода и параметры (аргументы) метода. Например, у метода `run()` — сигнатура `run()`, модификатор доступа — `public`, возвращаемое значение — `void` (ничего не возвращает). У метода `calcDistance()` — сигнатура `calcDistance(double hours)`, возвращаемое значение типа `double`, модификатор доступа — `public`. Телом метода называется блок кода, заключённый в фигурные скобки, определяющие область действия метода. Если объединить вместе сигнатуру, модификатор и возвращаемое значение, то это будет называться **контракт метода**.

Часто при изучении ООП возникает проблема с пониманием метода как абстракции, метода как описания последовательности команд, которые могут выполниться в случае его вызова. Например, не всегда есть чёткое понимание, когда же выполняется содержимое тела метода. Метод выполняется в момент вызова, например, в некотором методе `main()`. Но при этом метод может вообще никогда не выполниться, если никто его не вызовет или не создаст экземпляр класса, его содержащий.

Возвращаемыми значениями метода могут быть следующие.

1. Метод может вернуть значение любого из примитивных типов (`int`, `char`, `double` и т. д.).
2. Метод может вернуть объект, т. е. экземпляр класса, текст типа `String` или массив.
3. Метод может ничего не возвращать, тогда вместо конкретного типа указывается ключевое слово `void`. В этом случае метод работает как набор команд, которые в нём выполняются. Например, вывод на печать, обработка данных, отправка сообщения по сети и т. д.

Возвращаемое значение метода представляет собой результат работы метода. Если в методе указано, что метод должен возвращать какое-либо значение, то в теле метода необходима инструкция вида `return результат`, где результат должен соответствовать типу возвращаемого значения. Так, если указано, что метод возвращает тип int, то и `return` должен возвращать `int`, а не `String` или `double`.

```
public int summa(int a, int b) {
     int c=a+b;
     return c;
}
```

или

```
public String generateText(String language) {
     String text="";
     if (language=="russian") {
          text="Привет";
     }
     else {
          text="Hello";
     }
     return text;
}
```

Без инструкции `return` код не скомпилируется. Возможны конструкции, в которых метод возвращает изначально предопределённый результат, например, в различного рода вспомогательных методах.

```
public class CheckMethod {
     public boolean getOk() {
          return true;
     }
}
```

Если указано, что метод должен возвращать значение, то результатом работы должен быть обязательно возврат какого-либо значения заданного типа. В случае использования ветвлений, в которых неизвестно точно, выполнятся ли инструкции хотя бы одной из веток, код не скомпилируется, так как непонятно однозначно, вернёт ли метод результат.

```
public boolean dataTransferred(int x) {
     if (x>0) {
          return true;
     } else if (x==0)
          return false;
}
```

В этом случае не охвачено всё пространство возможных событий. Методу непонятно, что делать, если $x<0$. В этом случае можно:

- дописать финальное `else` с инструкцией `return`;
- в конце метода добавить оператор `return`, который будет возвращать результат, если ни одна из предыдущих инструкций не сработала;
- в области действия метода создать и инициализировать переменную, присвоить ей значение в результате выполнения инструкций и в конце метода вернуть инструкцией `return`.

Пример реализации третьего варианта.

```
public boolean dataTransferred(int x) {
     boolean result=false;
     if (x>0) {
          result=true;
     } else if (x==0)
          result=false;
return result;
}
```

Справочник

Модификаторы доступа определяют возможность обращения к полям и методам класса, самим классам, конструкторам класса. В Java имеется четыре вида доступа к полям: `public`, `private`, `protected`, без модификатора.

В рамках этого курса достаточно рассмотреть только три основных модификатора:

- `public`: имеется право общего доступа, всё что отмечено как `public` — доступно из любых других классов;
- `private`: доступно только самому классу, т. е. можно вызвать метод или получить доступ к полю только внутри самого класса.
- `protected`: предоставляет возможность классам-наследникам использовать методы и поля класса-родителя, а также всем классам в одном пакете.

Таким образом, если поле или метод должны быть доступны всем, то устанавливается модификатор `public`. Иначе лучше установить модификатор `private`. На первых порах изучения Java этих двух модификаторов будет более чем достаточно. Правильным считается все поля класса помечать как `private` в целях безопасности и отсутствия конфликтов при выполнении методов разных классов. В этом случае создаются специальные методы для доступа к таким полям, называемые «геттер» и «сеттер» (англ. getter и setter).

```
package com.itcube;
import static java.lang.System.out;
public class Man2 {
     private double salary;
     private String name;
     private int age;

     public double getSalary() {
          return salary;
     }
     public void setSalary(double salary) {
          this.salary=salary;
     }
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name=name;
     }
     public void setAge(int age){
          this.age=age;
     }
     public int getAge(){
          return this.age;
     }
}
class Man2TestDrive {
     public static void main(String[] args) {
```

```
            Man2 misha = new Man2();
            out.println("Hello!");
            misha.setName("Миша");
            misha.setSalary(30000.0);
            misha.setAge(30);
            out.println(misha.getName()+": age="+
misha.getAge());
            out.println(misha.getName()+": salary="+
misha.getSalary());
        }
    }
```

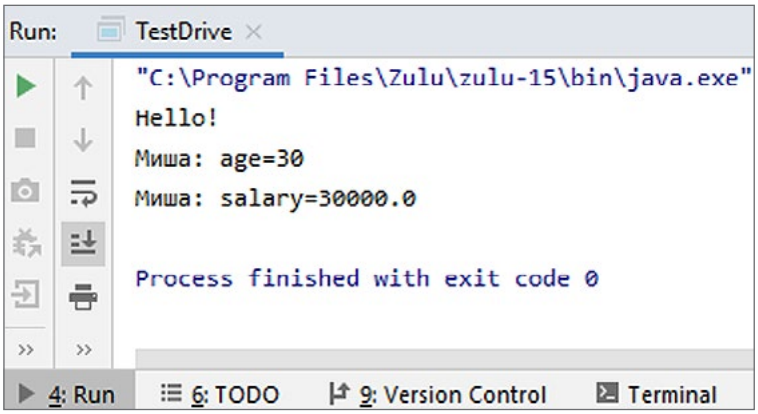

Рис. 47. Результат выполнения программы с геттерами и сеттерами

Особенностью классов является доступность методов одного класса друг другу независимо от типа доступа метода. Поля класса тоже доступны из любых методов этого класса независимо от их типа доступа. Исключения существуют только для вызовов нестатических методов и полей из статических методов — это будет рассматриваться в лабораторной работе № 5.2 «Статические элементы».

Так из любого обычного метода класса можно обращаться к любым другим методам этого же класса. Например, для класса `Wolf` можно модифицировать метод `hello()`, добавив в него вызовы методов `sound()` и `getAge()` следующим образом.

```
    public String hello() {
        if(getAge()>5){
            sound();
        }
        String s = name+" поздоровался";
        return s;
    }
```

Очевидно, что обычные методы других классов здесь так использовать не получится. Сначала необходимо создать экземпляр другого класса. После чего можно использовать, например, его public-методы. В случае со статическими элементами всё наоборот, это будет рассмотрено в лабораторной работе № 5.2 «Статические элементы».

Справочник

Параметром метода называется аргумент, принимаемый методом. Термин «аргумент» подразумевает, что конкретно и какому конкретно методу было передано, а параметр — в каком качестве метод применил это принятое. Параметры обозначаются в круглых скобках сразу после имени метода.

```
public void myMethod(int x, String s){
     x++;
     out.println(s+" увеличилось до "+x);
}
```

Метод использует значения (если это переменная примитивного типа) и ссылки (если объектного типа) для выполнения инструкций в теле метода. Параметрами метода могут быть не только переменные, но и массивы и списки.

```
public int summa(int[] elements){
     int result=0;
     for(int i=0; i<elements.length; i++){
          result=result+elements[i];
     }
     return result;
}
```

Особую роль в классе занимает конструктор.

Справочник

Конструктор — это особый метод, выполняемый при создании нового экземпляра класса. Одно из основных предназначений конструктора — инициализировать поля класса при его создании.

Имя конструктора всегда совпадает с именем класса, и у него отсутствует возвращаемый тип. Модификатор доступа может быть как `public` — тогда класс можно создать из любого места, так и `private` — класс нельзя создать ниоткуда, кроме как из самого экземпляра класса. Конструктор также может иметь параметры, как и любой метод. Для каждого класса по умолчанию создаётся пустой конструктор, необязательно прописывать конструктор отдельно.

```
public class Man{
     String name;
     //конструктор
     public Man(){
          name="Misha";
     }
}
```

```
public class Man{
     String name;
     //конструктор с параметром
     public Man(String name){
     //инициализация
          this.name=name;
     }
}
```

Важно!

Ключевое слово `this` означает, что переменная относится к полю класса. Оно используется при одинаковых именах переменных в полях класса и параметрах методов, чтобы не было путаницы, например при инициализации с помощью конструктора.

Тема конструкторов достаточно обширна и лишь слегка затрагивается в этом курсе.

Важно понимать, что разные переменные в классе имеют различную область видимости. Например, поля класса видны во всём классе: во всех методах, блоках, конструкторах, т. е. доступны из любого места класса. Переменные, объявленные в каком-то методе класса, доступны только в этом методе и недоступны из других методов класса или любого другого места класса.

Пакеты являются средством структурно-логического объединения кода. В процессе разработки они представляют собой всего лишь обычную папку в каталоге файлов. Если изучить структуру пакетов проекта в IDE IntelliJ, можно убедиться, что она полностью соответствует структуре папок при просмотре из проводника Windows.

Начиная с JDK 9 появились модули, что можно рассматривать как дальнейшее развитие идеи пакетов в сторону структурно-логической группировки кода (классов и ресурсов) как системы компонентов. Модули объединяют пакеты, классы и ресурсы. В результате они рассматриваются как часть целого и обращение к ним идёт по имени модуля.

Практическая часть

Цель работы: Ознакомление с основами объектно-ориентированного подхода в Java. Получение навыков работы с классами, объектами, полями и методами класса.

Ход лабораторной работы

1. Определить возвращаемый тип метода по его описанию. Здесь клиентом называется либо пользователь, вызывающий метод, либо некоторая программная структура (например, метод может вызываться из другого метода или класса).

Задача метода	Возвращаемый тип (примерный ответ)
Рассчитать корни квадратического уравнения	Массив из двух чисел типа `double`
Вывести сообщение «Hello, world!» в консоль	`void`
Сгенерировать сообщение «Hello, world!» и отправить на почту клиента	`void`
Сгенерировать сообщение «Hello, World!» в виде строки для дальнейшего использования	`String`
Сгенерировать сообщение «Hello, world!» в виде последовательности символов для дальнейшего использования	`char[]`
Сгенерировать сообщение «Hello, world!» и вывести его на консоль	`void`
Рассчитать результат сложения двух целых чисел и вывести его в консоль	`void`

Продолжение

Задача метода	Возвращаемый тип (примерный ответ)
Создать объект класса `Wolf` с заданными свойствами для дальнейшего использования	`Wolf`
Создать объект класса `Wolf` с заданными свойствами и вывести в консоль его свойства	`void`
Сгенерировать и вывести в консоль таблицу умножения	`void`
Рассчитать параметры 3D-модели дома	Зависит от типа результата (целые числа, вещественные, одно число или массив и т. д.)
Рассчитать параметры 3D-модели дома и отправить данные на сервер	`void`
Увеличить значение некоторого поля класса на единицу	`void`
Сгенерировать случайный символ для дальнейшего использования	`char`
Сгенерировать случайное значение типа `boolean`	`boolean`
Сгенерировать случайное значение типа `boolean` и передать его в метод `void m1(boolean b)`	`void`
Сгенерировать случайное значение типа `boolean` и передать его в метод `int m1(boolean b)`	`void`

2. Учитель берёт код класса `Bicycle` из документации Oracle по ссылке https://docs.oracle.com/javase/tutorial/java/concepts/class.html и удаляет возвращаемые типы методов. Проанализировать методы и расставить удалённые типы. Проанализировать код класса `BicycleDemo` по той же ссылке, разобрать последовательность выполнения методов, объяснить, почему все методы содержат ключевое слово `void`.

3. Запустить среду IntelliJ. При необходимости создать новый проект и класс с методом `main()`. Скопировать код классов `Bicycle` и `BicycleDemo`.

4. В классе `Bicycle` удалить метод `printStates()` и установить модификаторы доступа для всех полей класса как `private`. Запустив метод `main()` класса `BicycleDemo`, попробовать вывести в консоль те же значения, что и раньше. Убедиться, что необходимо добавить методы-геттеры в класс `Bicycle` для считывания значений полей.

```
public int getSpeed(){
    return speed;
}
```

Обсудить, почему геттеры должны возвращать конкретный тип, в данном случае `int`.

5. Заполнить таблицу для всех методов класса `Wolf`.

Метод	Сигнатура	Модификатор доступа	Возвращаемое значение
`run()`			
`sound()`			
…			

6. Создать класс `Tiger` со следующим кодом.

```
public class Tiger {
     private String name;
     private int age;
     public Tiger(String name) {
          this.name=name;
     }
     public String getName() {
          return name;
     }
     public int getAge() {
          return age;
     }
     public void setAge(int age) {
          this.age=age;
     }
     public void run(int km) {
          for (int i=0; i<km; i++) {
               out.println("Тигр пробежал "+i+" км.");
          }
     }
     public String voice(boolean day) {
          if (day) {
               return "Храп...";
          } else return "Рык...";
     }
     public static void main(String[] args) {
          Tiger tiger = new Tiger("Кубик");
          tiger.setAge(10);
          out.println(tiger.getName());
          tiger.run(3);
          out.println(tiger.voice(false));
          tiger.run(3);
          tiger.getName();
     }
}
```

7. Выполнить и проанализировать инструкции в методе `main()`.
8. Заполнить для класса `Tiger` такую же таблицу, как для класса `Wolf`.

9. Добавить следующие методы в код классов `Bicycle` и `Wolf`. Добавить вызовы и выполнить новые методы в методе `main()`. При необходимости добавить требующиеся новые поля класса (например, для метода `getMileage()` класса `Bicycle` требуется поле класса, хранящее стоимость).

Для класса `Bicycle`:

- Метод, возвращающий пробег велосипеда в заданной единице исчисления (км или мили).

Решение:

Добавить поле `int mileage`.

Добавить метод:

```
public double getMileage(String mes){
     if(mes=="km")
          return mileage/1.6;
     else
          return mileage;
}
```

- Метод, возвращающий последнее использованное значение параметра `decrement` метода `applyBrakes()`, т. е. с каким значением параметра последний раз вызывался метод `applyBrakes()` (добавить `private` поле класса для хранения значения `decrement` и изменить метод `applyBrakes()` для учёта значений).

Решение:

Добавить поле `int decr`.

Добавить метод:

```
public int getLastDecrement(){
     return decr;
}
```

- Метод, выводящий в консоль последнее значение параметра `decrement` метода `applyBrakes()`.

Решение:

Добавить метод:

```
public void printDecr(){
     out.println(getLastDecrement());
}
```

- Метод, возвращающий список использованных в методе `applyBrakes()` значений параметра `decrement`.

Решение:

Добавить поле `private ArrayList<Integer> decrements` для хранения значений decrement.

Добавить метод:

```
public ArrayList<Integer> getDecrements(){
     return decrements();
}
```

Если при тех же условиях нужен метод, возвращающий конкретное значение по порядковому номеру, то можно добавить метод:

```
public int getDecrements(int position){
     return decrements.get(position);
}
```

Также необходим метод, записывающий значения в динамический список:

```
public void setDecrements(int value){
     decrements.add(value);
}
```

- Метод, возвращающий объект — хозяина велосипеда.

Решение:
Добавить поле `Man owner`.
Добавить конструктор класса:

```
public Bicycle(Man){
     owner=man;
}
```

Добавить метод:

```
public Man getOwner(){
     return owner;
}
```

Если конструктор отсутствует, тогда необходимо добавить метод, задающий значение переменной `owner`, и проверку на существование объекта в методе `getOwner()`:

```
public Man getOwner(){
     if(owner!=null)
          return owner;
     else throw new NullPointerException();
}
```

Для класса `Wolf`:
- Метод, возвращающий родителя `Wolf`.

Решение:
Добавить поле класса типа `Wolf[]` и инициализировать это поле либо в месте объявления, либо в конструкторе, либо в методе, задающем значение этого поля.

```
Wolf[] parents = new Wolf[2];
public Wolf(Wolf mother, Wolf father){
     parents[0]=mother;
     parents[1]=father;
}
```

Здесь возможны различные варианты решения. Например, задать родителей в виде двух отдельных переменных типа `Wolf` или массив `Wolf[]` инициализировать в конструкторе.

Добавить следующий метод.

```
public Wolf getParent(int parentId){
    return parent;
}
```

- Метод, возвращающий родителей `Wolf`.

Решение:
Добавить поле класса, как в пункте 6.
Добавить метод:

```
public Wolf[] getParents(){
    return parents;
}
```

Предпочтительнее возвращать в виде массива.

- Метод, возвращающий всех родственников `Wolf` в виде `ArrayList`.

Решение:
Добавить поле класса в виде динамического массива `public ArrayList<Wolf> relatives=new ArrayList<>()` и инициализировать его в конструкторе.

```
public Wolf(){
    relatives.add(new Wolf("Mother");
    relatives.add(new Wolf("Father");
    // и т.д.
}
```

Добавить следующий метод.

```
public ArrayList<Wolf> getAll(){
    return relatives;
}
```

- Метод, задающий возраст `Wolf`.

Решение:

```
public void setWolfAge(int age){this.age=age;}
```

- Метод, выводящий возраст Wolf в консоль.

```
public void printWolfAge(){out.println(getWolfAge());}
```

Выводы: в ходе выполнения лабораторной работы были освоены базовые принципы работы с классами и объектами в Java.

Контрольные вопросы

1. Чем класс отличается от экземпляра этого класса (объекта)?
2. Для чего используется ключевое слово `new`?
3. Что такое конструктор? Может ли у класса не быть конструктора?
4. Что такое инициализация полей класса?
5. Как создать новый объект в Java-проекте IntelliJ?

6. Переменная объектного типа является ссылкой или значением?
7. Чем поля класса отличаются от методов?
8. Какие бывают модификаторы доступа и чем они различаются?
9. Входит ли область действия метода в область действия класса, которому этот метод принадлежит?
10. Может ли поле класса быть переменной объектного типа?
11. Могут ли два разных класса иметь методы с одинаковой сигнатурой?

Лабораторная работа № 5.2. Статические элементы

Теоретическая часть

Статические элементы играют важную роль при разработке программ на языке Java. Статическими бывают методы, поля, классы, инициализаторы, константы. Модификатор `static` (англ. «статичный», «постоянный») делает переменную или метод независимыми от объекта. Статические элементы определены на уровне классов, а не объектов, т. е. не привязаны к экземплярам классов, которые содержат статические элементы.

Важно!

Статические методы и поля не требуют наличия объекта класса, в котором содержатся. Обращаться к статическим полям и вызывать статические методы можно без создания экземпляра класса. Так, статические поля и методы вызываются лишь с указанием имени своего класса через точку.

```
Math.PI;
Math.abs(0);
```

Статические методы не связаны с нестатическими переменными и методами своего класса, потому что работа с нестатическими элементами требует наличия объекта класса. Можно сказать, что для статических методов их класс является лишь своего рода оболочкой, папкой, пространством имён, в котором они находятся. Это и является одной из весомых причин для их использования.

```
public class Statical {
    public static double myFormula(double x, double y) {
        return Math.pow(x, y);
    }
    public static void printSomeText(String text) {
        System.out.println(text);
    }
    static double calcIMT(double ves, double rost) {
        //расчёт индекса массы тела
        double imt=ves/(Math.pow(rost,2));
        return imt;
    }
}
class TestDrive{
    public static void main(String[] args) {
        System.out.println(Statical.calcIMT(100,1.87));
    }
}
```

Частая ошибка — попытка внутри класса вызвать статические методы из нестатических (например, метода `main()`), не создав заранее объект самого класса. Это невозможно, так как статические методы и поля класса находятся вне контекста самого объекта класса. Поэтому нужно, например, в методе `main()` сначала создать объект класса и только потом вызывать его нестатические методы или использовать нестатические поля.

```
package com.itcube.oop;
public class Statical2 {
     public void nonstaticPrint(String name) {
          System.out.println("Hello, "+name);
     }
     public static void staticPrint(String name) {
          System.out.println("Hello, "+name);
     }
     public static void main(String[] args) {
          staticPrint("Исидор");//ok
          nonStaticPrint("Исидор");//ошибка!
          Statical2 st = new Statical2();
          st.nonstaticPrint("Исидор");//ok
     }
}
```

При наведении на красный маркер возникает ошибка.

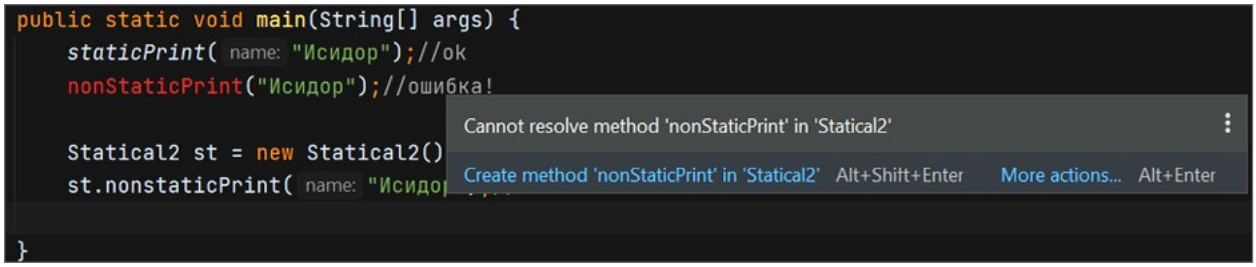

Рис. 48. Ошибка при вызове нестатического метода из статического

Таким образом, статические методы вызываются без создания экземпляра класса, а нестатические вызываются у экземпляра класса.

В классе Math из лабораторной работы № 2 «Переменные. Операторы» большинство методов являются статическими. Например, `Math.pow(x,y)`, `Math.sin(x)` и т. д.

Нестатическое поле класса инициализируется и принадлежит конкретному экземпляру класса после создания этого экземпляра и выделения памяти в куче. Статические же поля, так же, как и методы, определены на уровне класса, а не объекта, поэтому являются общими для всех экземпляров класса. Статические поля можно представить как своего рода глобальные переменные, общие для класса. Одним из типовых применений статических полей является их использование как счётчиков.

```
public class Pupil {
     private static int counter=1;
     private int id;
     private String name;
     public static final String school="131";
     private Pupil() {
```

```
        id=counter++; }
    private Pupil(String name) {
        this.name=name;
        id=counter++;
    }
    public int getId() {
        return id;
    }
    public static void main(String[] args) {
        Pupil vasya = new Pupil("Вася");
        out.println("vasya id=" + vasya.getId());
        Pupil petya = new Pupil("Петя");
        out.println("petya id="+petya.getId());
        //безымянный ученик
            out.println(new Pupil().getId());
    }
}
```

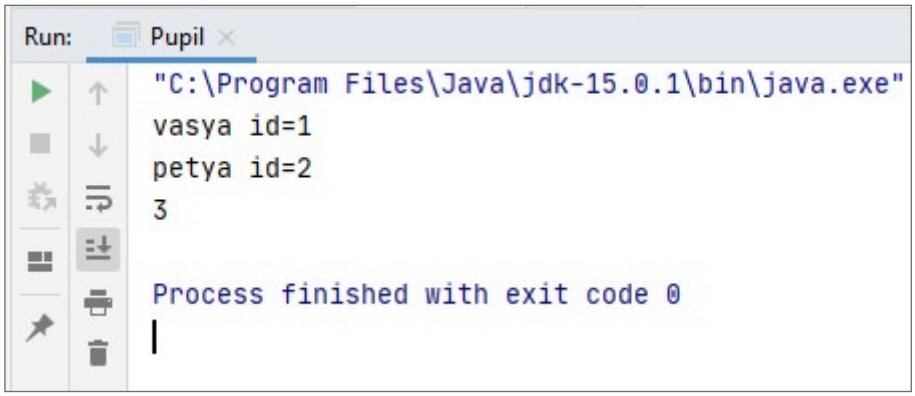

Рис. 49. Результат применения статической переменной в качестве счётчика

Если статическое поле имеет модификатор `public`, тогда оно доступно из любого места и вызывается аналогично методу через имя своего класса и точку.

```
out.println(Pupil.school);
```

Практическая часть

Цель работы: освоить работу со статическими элементами в Java.

Ход лабораторной работы

1. Запустить среду IntelliJ. При необходимости создать новый проект и класс с методом `main()`. Создать класс `Statical2`, реализовать код класса и вызвать его метод в методе `main()` класса `TestDrive`. Проанализировать результаты.

2. В открытых источниках найти информацию про градации значений коэффициента индекса массы тела и написать программу для расчёта индекса массы тела и вывода соответствующих рекомендаций. Создать класс, метод расчёта и проверку реализовать в статических методах. Использовать метод `calcIMT` класса `Statical` (необходимо дописать метод, выдающий рекомендацию в зависимости от полученного значения коэффициента). Выполнить код.

Примерный код нового метода может быть следующим.

```
static String recommendationIMT(double IMT) {
     if (IMT<18.5) {
          return "Ниже нормы";
     } else if ("IMT>=18.5"&&"IMT<25") {
          return "Нормальный вес";
     } else if ("IMT>=25"&&"IMT<30") {
          return "Избыточный вес";
     } else {
          return "Не удалось рассчитать";
     }
}
```

3. Реализовать код класса `Pupil`. Выполнить и проанализировать метод `main()`.

4. Разобрать использование статических полей на примере класса `Pupil`. Убедиться, что статическое поле является общим для всех экземпляров класса `Pupil`.

5. Реализовать счётчик на базе статического поля типа `int` для любого из классов из предыдущих лабораторных работ. Проверить работу в методе `main()` (создать несколько экземпляров, проверить значения счётчика и убедиться, что он возрастает при создании каждого нового экземпляра).

6. Создать новый класс `MyMath` со следующим методом.

```
public double abs(double a) {
     if (a < 0)
          return a*-1
     else
          return a;
}
```

В любом методе `main()` проверить этот метод и сравнить вызов с методом `abs()` класса `Math`. Убедиться, что для вызова статического метода `Math.abs()` не нужен экземпляр его содержащего класса в отличие от метода `abs()` класса `MyMath`.

7. Создать в классе `Statical2` любые нестатические поля. Попробовать вызвать их в статическом методе. Убедиться, что возникает ошибка, проанализировать ситуацию.

Выводы: в ходе выполнения лабораторной работы освоена работа с некоторыми статическими элементами в Java. Получены первичные представления о статических элементах и их применении.

Контрольные вопросы

1. Для чего нужны статические элементы?
2. Чем статические поля отличаются от статических методов?
3. Можно ли из статического метода класса вызвать нестатический метод класса?
4. Можно ли из нестатического метода класса вызвать статический метод класса?
5. Можно ли из статического метода класса вызвать статический метод класса?
6. Можно ли из статического метода класса обратиться к статическому полю класса?
7. Для чего нужны статические поля класса?

Лабораторная работа № 6. Управляющие структуры. Циклы

Теоретическая часть

Одним из видов управляющих структур в программировании являются циклы.

Справочник

Циклы используются для организации многократно повторяющихся блоков инструкций.

На самом деле в программировании можно обойтись и без циклов, реализовав алгоритм с помощью последовательных инструкций. Однако могут возникнуть сложности, если необходимо выполнить определённый блок инструкций 10, 100, 1000 и более раз. Например, если есть необходимость обойти большое игровое поле и в каждой ячейке установить ту или иную текстуру (трава, песок, лужа) и далее установить объекты (танк, автомобиль, мотоцикл), то намного проще это сделать, использовав циклы. Особенно если представить, что ячеек на поле может быть 10 000. Намного проще это сделать в цикле и один раз прописать все инструкции, чем 10 000 раз писать последовательно один и тот же код.

Цикл `for()` выглядит следующим образом.

```
for (int i=0; i<10; i++){
     инструкция 1;
     инструкция 2;
     …
     инструкция n;
}
```

Здесь целочисленная переменная `i` является счётчиком, который принимает значения от 0 до 9 с шагом 1. Область кода с инструкциями внутри фигурных скобок оператора `for()` называется телом цикла.

Перед первым выполнением тела цикла значение `i` равно нулю. После первого выполнения тела цикла (иначе, после первой итерации цикла) поток управления программой переходит на начало цикла. Происходит операция `i++` и значение `i` становится равным 1. После чего следует вторая итерация, i снова увеличивается на 1, становится равным 2, следует третья итерация и т. д. С каждой итерацией цикл увеличивает значение `i` на 1 и выполняет все инструкции в теле цикла. Так происходит до тех пор, пока верно условие `i < 10`. После 10-й итерации `i` принимает значение 10, условие перестаёт выполняться и происходит выход из цикла.

Важно понимать, что в данном случае областью видимости переменной `i` является только тело цикла, вне цикла она недоступна. Если бы переменная `i` была объявлена перед циклом, тогда была бы доступна как в цикле, так и вне его.

Другой вариант записи цикла:

```
for (int i=9; i>=0; i--){
     инструкция 1;
     инструкция 2;
     …
     инструкция n;
}
```

Можно организовать прохождение цикла не только в сторону увеличения значений счётчика, но и в сторону уменьшения. В примере выше значение счётчика равно 9, и после каждой итерации оно уменьшается на 1. Цикл будет выполняться до тех пор, пока *i* не примет значения -1. В этом случае цикл завершится.

Шаг изменений значений счётчика может быть равен не только 1, но и 2, 3 и т. д.

```
for (int i=0; i<1000; i+=2){
     инструкция 1;
     инструкция 2;
     …
}
```

Значения счётчика цикла часто используются в теле цикла `for()` при выполнении инструкций.

Пример реализации сложения чисел от 0 до 99, где с помощью счётчика осуществляется наращение суммы.

```
public class Simple2 {
     public static void main(String[] args) {
          int summa=0;
          for (int i=0; i<100; i++) {
               summa=summa+i;
          }
          out.println(summa);
     }
}
```

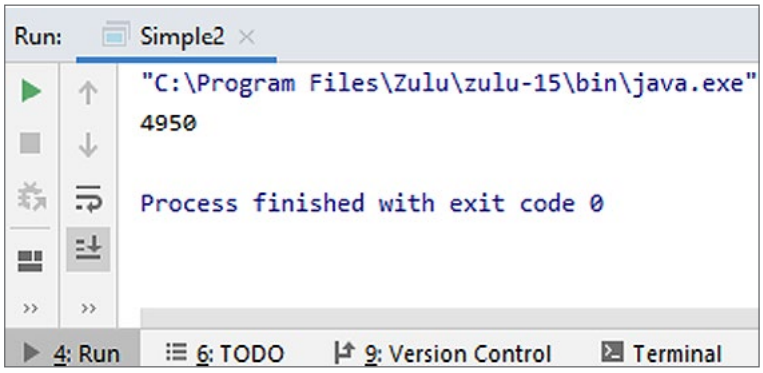

Рис. 50. Результат суммирования чисел от 1 до 100

Цикл `for()` может содержать вложенные циклы `for()`. В этом случае он будет называться двойным циклом, тройным циклом и т. д. Например, следующим образом можно вывести в консоль таблицу умножения от 1 до 10.

```
public static void main(String[] args) {
     for (int i=0; i<10; i++) {
          for (int j=0; j<10; j++) {
               out.print((i+1)*(j+1)+"\t");
          }
          out.println();
     }
}
```

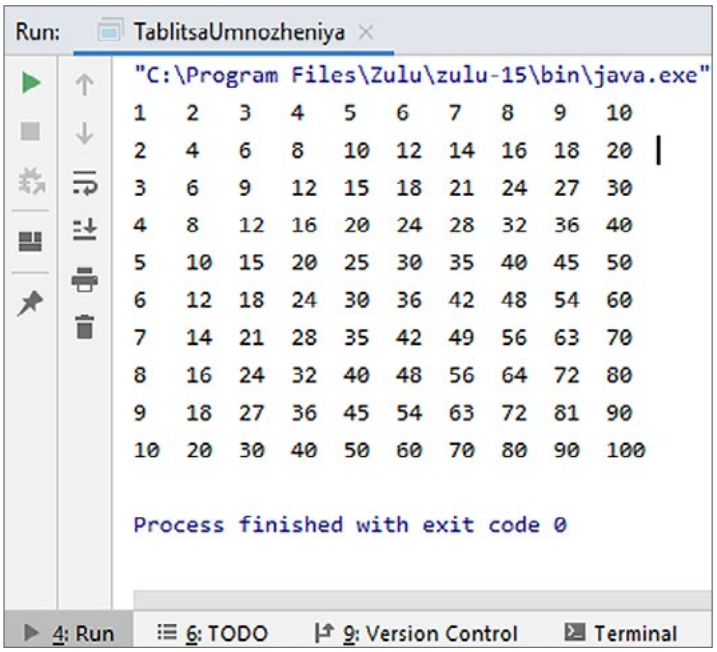

Рис. 51. Вывод таблицы умножения с помощью двойного цикла

Отдельный интерес представляет оператор `continue`, с помощью которого можно сразу перейти к следующей итерации цикла.

```
for (int i=0; i<10; i++) {
     for (int j = 0; j<10; j++) {
          if (i<=j)
          out.print((i+1)*(j+1)+"\t");
          else
               continue;
     }
     out.println();
}
```

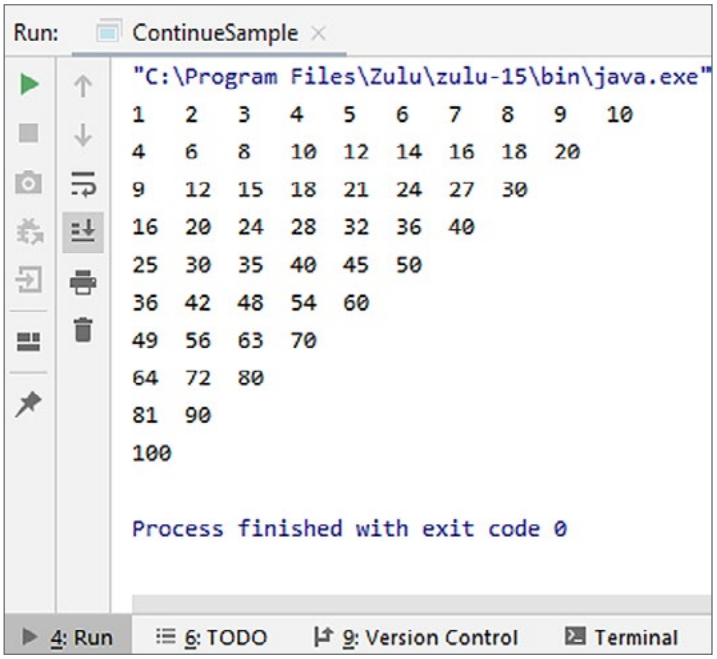

Рис. 52. Результат работы цикла с использованием оператора `continue`

По результату видно, что как только перестаёт соблюдаться условие $i \leqslant j$, текущая итерация завершается, значение счётчика увеличивается на 1, тело цикла начинает выполняться заново (при соблюдении условия цикла).

Стоит отметить, что оператор `break` действует одинаково как в условиях, так и в циклах и означает досрочный выход. Например, подобное условие завершит цикл, как только `i` станет равным 6.

```
for (int i=0; i<10; i++) {
     if (i>5)
          break;
}
```

Другая удобная форма организации циклов — цикл `while()` (пока).

Важно!

Всё, что можно выполнить на основе `for()`, также можно реализовать и посредством цикла `while()`.

Разница в том, что в `for()` начальные и конечные значения счётчика вместе с шагом задаются в явной форме. В `while()` же количество итераций контролируется с помощью условия, прописанного в круглых скобках. Когда условие перестаёт выполняться, происходит завершение цикла.

Таким образом, при необходимости выполнить цикл заданное количество раз проще использовать цикл `for()`, а при необходимости повторения действий в зависимости от логического выражения удобнее использовать цикл `while()` для организации циклов.

Пример цикла `while()`, подсчитывающего сумму числе от 1 до 100, эквивалентного циклу `for()`.

```
int summa=0;
int i=0;
while (i<100) {
     summa=summa+i;
     i++;
}
out.println(summa);
```

В этом случае реализация счётчика такова: счётчик инициализируется перед циклом, условие завершения записывается в круглых скобках оператора `while()`, а инкрементируется счётчик в теле цикла.

В следующем примере применение именно `while()` более оправдано.

```
boolean isOk = true;
int number=0;
while (isOk) {
     number+=10;
     if (number>100) {
          isOk=false;
     }
}
```

Здесь в цикле `while()` проверяется значение логической переменной *isOk*. Если условие истинно, то следует выполнение тела цикла. Цикл завершается, когда *isOk* становится равной false.

Для организации бесконечного цикла `for()` используется инструкция `for(;;)`.

```
for (;;) {
    out.println("Привет, Куб!");
}
```

Данный цикл будет выполняться бесконечно, и для его остановки необходимо нажать на значок с красным квадратом на панели навигации редактора IntelliJ.

Рис. 53. Панель навигации редактора IntelliJ

Также можно заложить условие досрочного завершения цикла посредством ключевого слова `break`. Например, у нас есть задача выполнять цикл в течение какого-то времени.

```
for (;;) {
    long startTime=System.nanoTime();
    out.println("Привет, Куб!");
    out.println(System.nanoTime());
    if (System.nanoTime()-startTime>1000000) {
        break;
    }
}
```

Практическая часть

Цель работы: ознакомиться с циклами в Java и научиться работать с ними.

Ход лабораторной работы

1. Запустить среду IntelliJ. При необходимости создать новый проект и класс с методом `main()`. Выполнить все примеры из теоретической части.

2. Создать класс `TestPrivet`. В методе `main()` объявить переменные `String name="IT Cube"` и `String text="Привет"`. Реализовать и выполнить код, выводящий в консоль текст «Привет, IT Cube» 100 раз:

- с помощью цикла `for()` со счётчиком `i` от 0 до 100;
- с помощью цикла `for()` со счётчиком `i` от 100 до 0;
- с помощью цикла `for()` со счётчиком `i` от 200 до 0;
- с помощью цикла `for()` со счётчиком `i` от 0 до 300;
- с помощью цикла `while()`;
- *с помощью конструкции `do{} while()`;
- с помощью бесконечного цикла `for(;;)`;
- с помощью цикла `for()` со счётчиком `i` от 0 до 10;
- с помощью двойного цикла `for()` со счётчиком *i* от 0 до 10 и счётчиком `j` от 0 до 10;
- с помощью тройного цикла `for()` со счётчиками на ваше усмотрение.

3. Перенести все отдельные реализации вывода в консоль из предыдущего пункта в методы, т. е. «упаковать» их в методы, например, следующим образом.

```
public static void print1(String text, String name){
    for(int i=0; i<100; i++){
        System.out.println(text+", "+name);
    }
}
```

4. Переписать метод `main()`. Выполнить ранее созданный код только через вызов методов.
5. Обсудить удобство инкапсуляции (упаковки данных и методов в единый компонент).
6. Обсудить варианты использования `for()` и `while()`.
7. Написать код для вычисления факториала заданного числа с помощью цикла `for()`.
8. Написать код для вычисления факториала заданного числа с помощью цикла `while()`.
9. Написать код для расчёта среднего арифметического 2500 случайных целых чисел, используя цикл `for()` или `while()`. Случайные числа генерировать посредством классов `Random` или `Math`.
10.* Реализовать алгоритм сортировки «пузырьком».

Выводы: в ходе выполнения лабораторной работы получено базовое представление об управляющих структурах языка Java.

Контрольные вопросы

1. Что означают фигурные скобки у операторов `if()`, `for()`, `while()`?
2. Если внутри оператора `if()` находится другой оператор `if()`, входит ли внутренний `if()` в область видимости внешнего?
3. За счёт чего можно сделать цикл `for()` бесконечным?
4. За счёт чего можно организовать ветвление в программе?
5. Что будет, если счётчику в цикле `for()` не присвоить значение?
6. Можно ли внутри одного цикла `for()` разместить другой цикл `for()`?
7. Как можно использовать конструкцию `for(;i<10;){...}`?

Лабораторная работа № 7. Массивы

Теоретическая часть

Для реализации многих алгоритмов в языках программирования недостаточно только переменных. В переменной хранится одно значение, но часто есть потребность хранить и обрабатывать большие объёмы данных. В этом помогают массивы.

Справочник

Массив — это структура данных, хранящая набор значений (элементов массива), обозначающихся индексом или набором индексов.

В Java массив является объектом.

Важно!

Массивы в Java в отличие от ряда других языков (Visual Basic, Python) не являются динамически расширяемыми, т. е. необходимо при создании массива сразу указать количество его элементов.

Условно массивы относятся к единичным переменным так же, как циклы относятся к последовательному коду. Можно вообще не использовать массивы, а использовать только обычные переменные. Но тогда может возникнуть ситуация, в которой будет необходимо инициализировать тысячи переменных. Например, в вышеописанной ситуации с игровым полем, состоящим из 10000 ячеек. Допустим, что каждой ячейке должно соответ-

ствовать числовое значение, и эти значения должны постоянно изменяться. Где хранить подобные настройки? Действительно, можно создать 10 000 переменных и разместить значения в них. Но будет намного эффективнее использовать одну структуру данных и хранить все данные в ней. Для этого очень хорошо подходят массивы.

Для создания массива из десяти целых чисел необходимо объявить массив чисел целого типа с помощью конструкции `int[]` и затем создать объект массива.

```
int[] numbers;
numbers = new int[10];
```

Можно объединить обе операции в одну.

```
int[] numbers = new int[10];
```

Эквивалентной формой записи будет следующая.

```
int numbers[] = new int[10];
```

Аналогично можно объявить массив любых других типов.

```
double dnumbers[] = new double[100];
boolean oks[] = new boolean[5];
```

Доступ к элементу массива осуществляется посредством индекса — порядкового номера элемента, заключённого в квадратные скобки: `numbers[5]`, `dnumbers[99]`, `oks[4]`.

Важно!

Индексация элементов массива в Java начинается с 0. Поэтому первый элемент массива имеет индекс 0, второй — индекс 1, третий — индекс 2, а индекс последнего равен количеству элементов массива минус 1.

Так, если объявлен и создан массив `int[] numbers = new int[10]`, значит он содержит элементы с индексами 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Если обратиться к элементу `numbers[10]`, произойдёт ошибка и программа досрочно завершится с соответствующим сообщением в консоли, так как такого элемента не существует.

Для записи значений в элементы массива применяется обычное присваивание конкретному элементу нужного значения:

```
numbers[0]=5;
numbers[1]=6;
numbers[2]=-1;
```

Легко догадаться, что циклы `for()` очень удобны для заполнения массивов. В поле `length` хранится значение количества элементов массива, соответственно её можно использовать в условии цикла.

```
public static void main(String[] args) {
    boolean[] oks = new boolean[100000];
    Random random = new Random();
    for (int i=0; i<oks.length; i++) {
        oks[i]=random.nextBoolean();
        System.out.println("boolean["+i+"]="+oks[i]+";");
    }
}
```

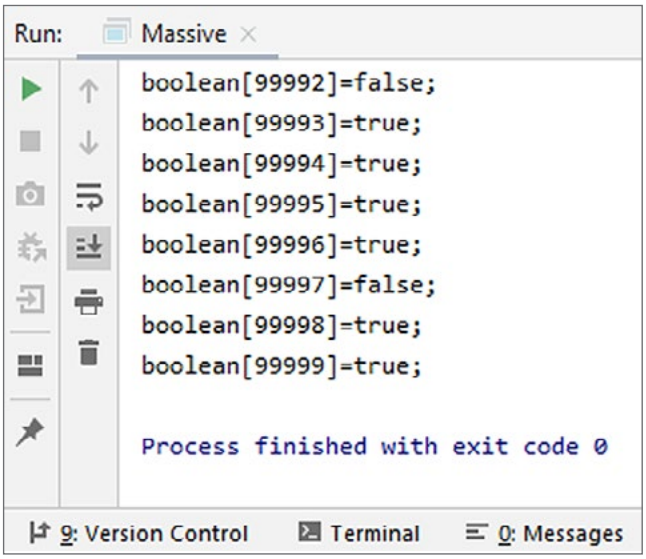

Рис. 54. Результат заполнения массива в цикле

Другой вариант заполнить массив значениями — изначальная инициализация, т. е. при объявлении:

```
int[] numbers={0, 2, 1, 3, 100};
String[] texts={"мама", "мыла", "раму"};
```

Важно!

По умолчанию в Java при создании массива элементам присваивается значение 0, т. е. «пустой» массив будет массивом из нулей.

Кроме одномерных массивов, можно создавать двумерные и трёхмерные массивы.

```
int[][] nums = new int[10][10];
```

Работа с двумерными массивами сложнее, так как на самом деле размерность второго измерения можно не указывать, а объявлять отдельно для каждого значения первого измерения через одномерные массивы.

```
int[][] nums = new int[10][];
```

Здесь рассматривается простой вариант создания двумерного массива, а именно, когда размерности изначально указаны. Работа с двумерным массивом не отличается от таковой в случае одномерного, например, при обращении к элементам. При этом правило индексации массива с 0-го элемента также сохраняется.

```
nums[0][0]=5;
System.out.println(nums[9][9]);
```

Чтобы заполнить двумерный массив, удобно использовать двойной цикл `for()`.

```
for (int i=0; i<10; i++) {
     for (int j=0; j<10; j++) {
          nums[i][j]=random.nextInt(100);
     }
}
```

К двумерным массивам тоже применима инициализация при объявлении.

```
int[][] nums={ {0,1,2}, {3,4,5}, {6,7,8}}; }
```

Основные задания данной лабораторной работы представляют собой задачи, развивающие логику и алгоритмическое мышление. Во всех заданиях требуется вывод в консоль значения двумерных массивов в виде матрицы. Все массивы заполняются комбинацией нулей и единиц.

```
package com.itcube.matrix;
import static java.lang.System.out;
public class Matrix1 {
    public static void main(String[] args) {
        int[][] a = new int[10][10];
        for (int i=0; i<10; i++) {
            for (int j=0; j<10; j++) {
                out.print(a[i][j]+" ");
            }
            out.println();
        }
    }
}
```

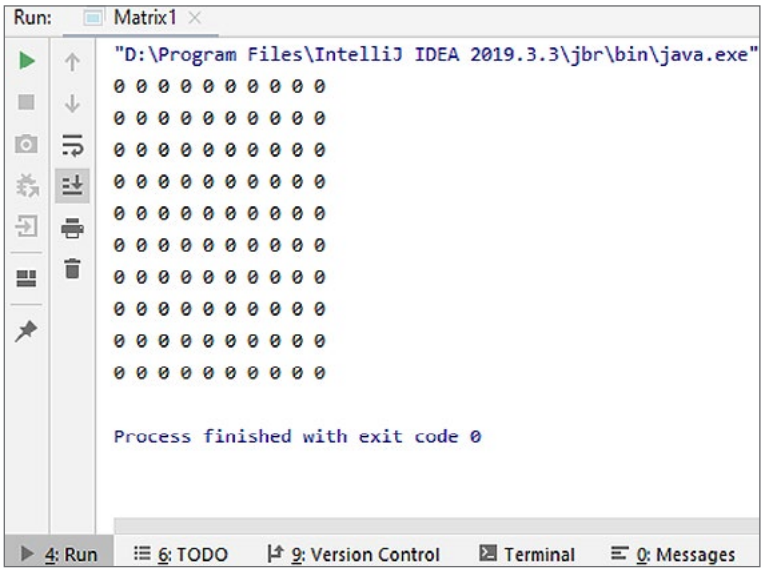

Рис. 55. Вывод пустого двумерного массива целых чисел в консоль

С точки зрения чистоты кода лучше не задавать ограничения счётчиков цикла `for()` в качестве чисел, а передавать их как переменные или параметры метода. Например, вместо числа 10 указать переменную `k`, предварительно инициализировав её перед циклом.

С точки зрения хранения данных нет разницы, работать ли с двумерным массивом размерности 10×10 или с одномерным массивом размерности 100. Для этого можно перейти на индексацию вида $i \cdot m + j$.

Обращение к элементам матрицы, представленной в виде двумерного массива:

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]
a[4][0]	a[4][1]	a[4][2]	a[4][3]	a[4][4]

Обращение к элементам матрицы, представленной в виде одномерного массива:

a[0]	a[1]	a[2]	a[3]	a[4]
a[5]	a[6]	a[7]	a[8]	a[9]
a[10]	a[11]	a[12]	a[13]	a[14]
a[15]	a[16]	a[17]	a[18]	a[19]
a[20]	a[21]	a[22]	a[23]	a[24]

Каждый элемент такой матрицы находится на пересечении *i*-й строки и *j*-го столбца. Например, `a[12]` на пересечении 2-й строки и 2-го столбца; `a[0]` — на пересечении 0-й строки и 0-го столбца; `a[17]` — 3-й строки и 2-го столбца и т. д. Тогда можно применять универсальную формулу для обращения через индекс массива к элементам подобной таблицы: `a[i * m+j]`, где i — номер строки, j — номер столбца, m — количество столбцов таблицы. Например: `a[12]` — это `a[2 * 5+2]`; `a[0]` — это `a[0 * 5+0]`, `a[17]` — это `a[3 * 5+2]` и т. д.

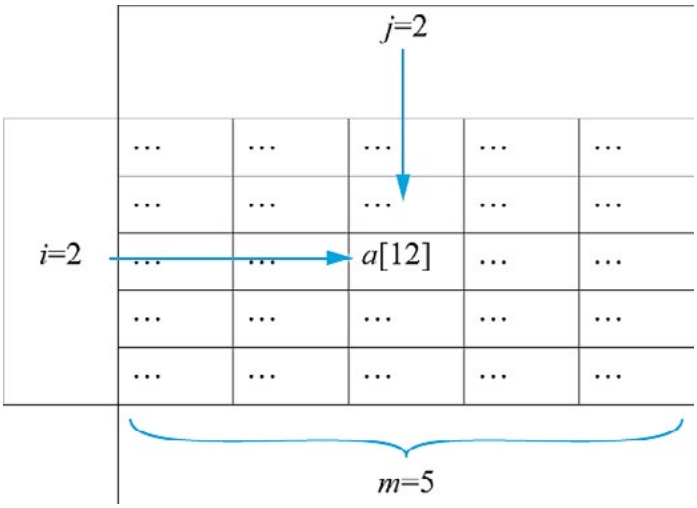

Рис. 56. Расположение элемента `a[12]`

Таким образом, одномерный массив можно представить в виде таблицы и соответственно к нему обращаться. Образно можно представить такую таблицу в виде ленты, которую разрезали и представили в виде *n* кусочков по *m* ячеек.

a[0]	a[1]	a[2]	a[3]	a[4]	…	…	…	a[20]	a[21]	a[22]	a[23]	a[24]

Для выполнения данной лабораторной работы необходимо усвоить материал по управляющим структурам, представленный в лабораторной работе № 6 «Управляющие структуры. Циклы». Также может пригодиться таблица логических операций.

Практическая часть

Цель работы: закрепление умений работы с управляющими структурами в Java с помощью написания программ обработки массивов.

Ход лабораторной работы

1. Вывести в консоль матрицу 10×10 из нулей, используя только двойной цикл `for()` и команду `out.println` (без использования массива `int[][]`).

Примерный код может быть следующим.

```
for (int i=0; i<10; i++) {
     for (int j=0; j<10; j++) {
          out.print(0+" ");
     }
          out.println();
}
```

Обратите внимание, что во внутреннем цикле используется именно `out.print()`, а в конце внешнего цикла — `out.println()`, поскольку необходим построчный вывод. Также необходимо понять смысл использования пробела в кавычках в выражении `out.print(0+" ")` — это элемент форматирования для вывода чисел в строке через пробел.

2. Разобрать и реализовать все примеры из теоретической части.

3. Реализовать алгоритм, заполняющий нечётные строки матрицы нулями, а чётные — единицами. Вынести код из метода `main()` и оформить его в виде статических методов. При этом будут решены сразу две задачи: приведение кода к удобному виду и рефакторинг — превращение спагетти-кода кода в более структурированный.

Дополнительно можно рассказать ученикам:

- о терминах «спагетти-код» и «рефакторинг»;
- о понятии чистого кода и почему это важно;
- повторно о правильном именовании переменных и верблюжьем стиле (camelStyle).

Примерный код может быть следующим.

```
public class Main {
     public static void zeroes_ones() {
     int[][] a = new int[10][10];
     for (int i=0; i<10; i++) {
          for (int j=0; j<10; j++) {
               if (i % 2 == 0)
                    a[i][j]=0;
               else
                    a[i][j]=1;
          }
     }
          printAll(a);
}
public static void printAll(int[][] a) {
     for (int i=0; i<10; i++) {
          for (int j=0; j<10; j++) {
               out.print(a[i][j]+" ");
          }
```

```
            out.println();
        }
    }
    public static void main(String[] args) {
        zeroes_ones();
        }
    }
```

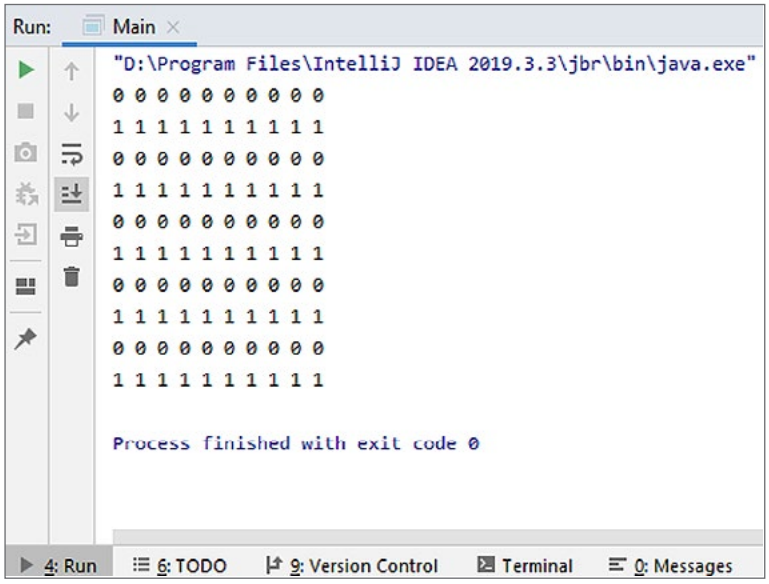

Рис. 57. Результат работы алгоритма

4. Реализовать алгоритм «Шахматная доска»: вывести в консоль матрицу в виде шахматной доски, в которой нули — белые клетки, а единицы — чёрные.

Примерный код может быть следующим.

```
public static void chess() {
    int[][] a = new int[10][10];
    for (int i=0; i<10; i++) {
        for (int j=0; j<10; j++) {
            if (j%2 != i%2 )
                a[i][j]=1;
            else
                a[i][j]=0;
        }
    }
    printAll(a);
}
```

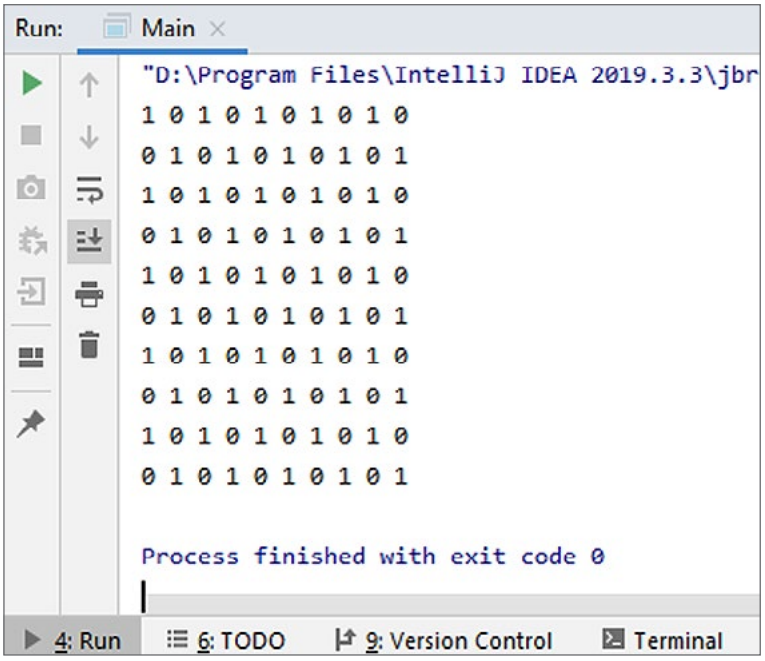

Рис. 58. Результат работы алгоритма «Шахматная доска»

Существует второй вариант основного условия для решения данной задачи:

```
if ((i+j)%2==1 )
     a[i][j]=1;
else
     a[i][j]=0;
```

Данное задание можно решить и более сложным способом, через комбинацию логических операторов `&&, ||`. Это решение менее элегантное, но имеет место быть.

6. Реализовать алгоритм «Главная диагональ»: вывести в консоль матрицу, заполненную единицами над главной диагональю (из левого верхнего в правый нижний угол) и нулями под ней.

Примерный код может быть следующим.

```
public static void diag1() {
     int[][] a = new int[10][10];
     for (int i=0; i<10; i++) {
          for (int j=0; j<10; j++) {
               if (i>=j)
                    a[i][j]=0;
               else
                    a[i][j]=1;
          }
     }
     printAll(a);
}
```

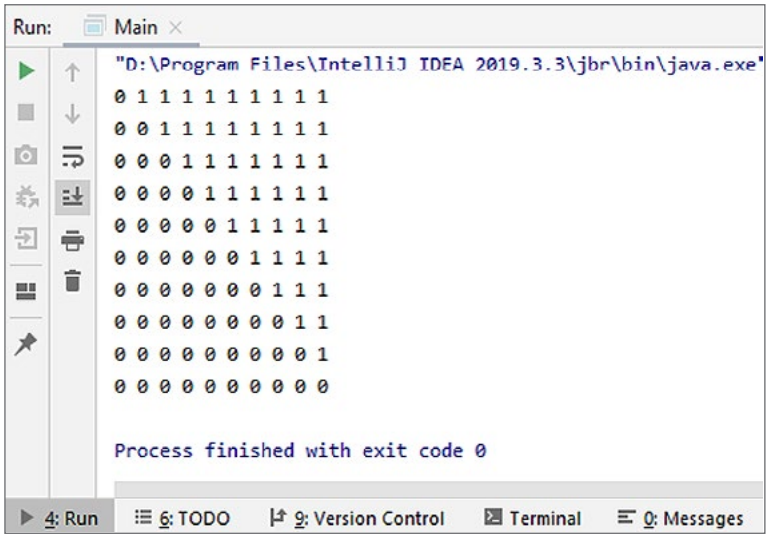

Рис. 59. Результат работы алгоритма «Главная диагональ»

7. Реализовать алгоритм «Побочная диагональ»: вывести в консоль матрицу, заполненную единицами под побочной диагональю (из правого верхнего в левый нижний угол) и нулями над ней.

Примерный код может быть следующим.

```
public static void diagPoboch() {
     int[][] a = new int[10][10];
     for (int i=0; i<10; i++) {
          for (int j=0; j<10; j++) {
               if (i+j<10 )
                    a[i][j]=0;
               else
                    a[i][j]=1;
          }
     }
     printAll(a);
}
```

Каждую из подобных задач обычно можно решить несколькими способами: либо детально задавать условие с помощью логического выражения, либо выявить закономерность расположения нулей и единиц и использовать краткую формулу.

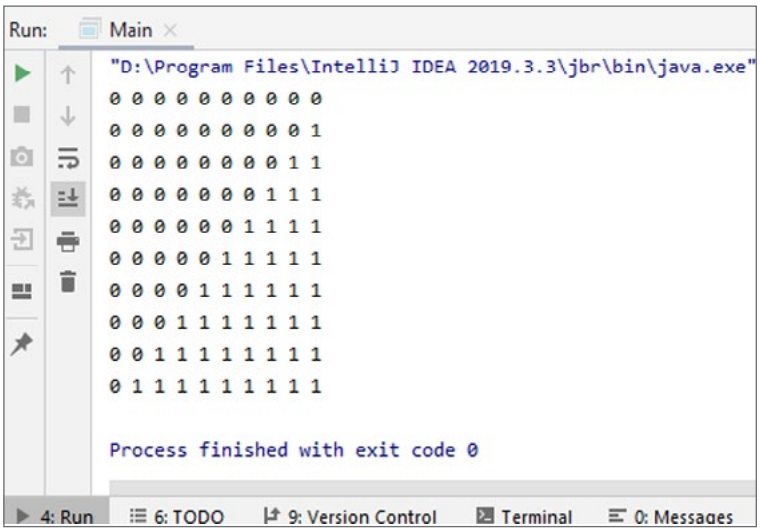

Рис. 60. Результат работы алгоритма «Побочная диагональ»

8. Реализовать алгоритм «Буква X»: вывести в консоль матрицу, изображающую букву X. Примерный код может быть следующим.

```
public static void xLetter(int num) {
     int[][] a = new int[num][num];
     for (int i=0; i<num; i++) {
          for (int j=0; j<num; j++) {
               if ((i==j)||(a.length-i-1)==j)
                    a[i][j]=0;
               else
                    a[i][j]=1;
          }
     }
     printAll(a);
}
public static void main(String[] args) {
     xLetter(10);
}
```

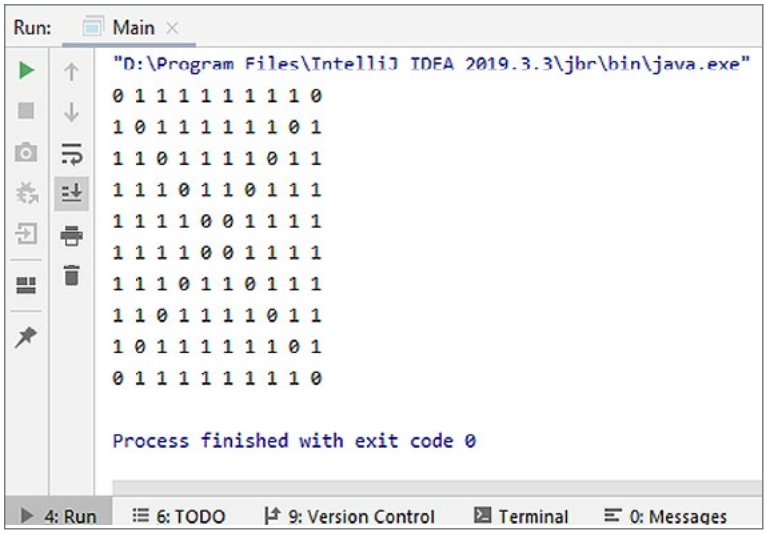

Рис. 61. Результат работы алгоритма «Буква X»

Дополнительный вопрос: почему в алгоритме из `a.length` вычитается 1?

9. Реализовать алгоритм «Песочные часы»: вывести в консоль матрицу, изображающую песочные часы.

Примерный код может быть следующим.

```
public static void sandWatches() {
     int[][] a = new int[10][10];
     for (int i=0; i<10; i++) {
          for (int j=0; j<10; j++) {
               if ((i<=j)&&(9-i)>=j||(i>=j)&&(9-i)<=j)
                    a[i][j]=0;
               else
                    a[i][j]=1;
          }
     }
     printAll(a);
}
```

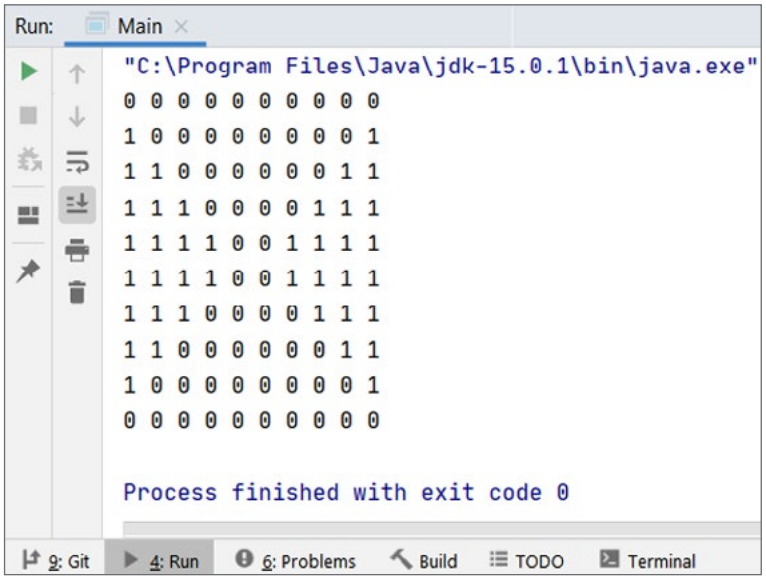

Рис. 62. Результат работы алгоритма «Песочные часы»

Дополнительный вопрос: как можно упростить следующее условие?

```
if ((i<=j)&&(9-i)>-=j||  (i>=j)&&(9-i)<=j)
```

Дополнительное задание: вывести инвертированные песочные часы.

10. Реализовать алгоритм «Заборы». Вывести в консоль матрицу, как на рисунке 56.

Примерный код может быть следующим.

```
public static void zabory() {
     int[][] a = new int[10][10];
     for (int k=0; k<10/2; k++) {
          for (int i=0+k; i<10-k; i++) {
               for (int j= 0+k; j<10-k; j++) {
                    if (k%2==0)
                         a[i][j] = 0;
               else
                         a[i][j]=1;
               }
```

```
            }
        }
        printAll(a);
    }
```

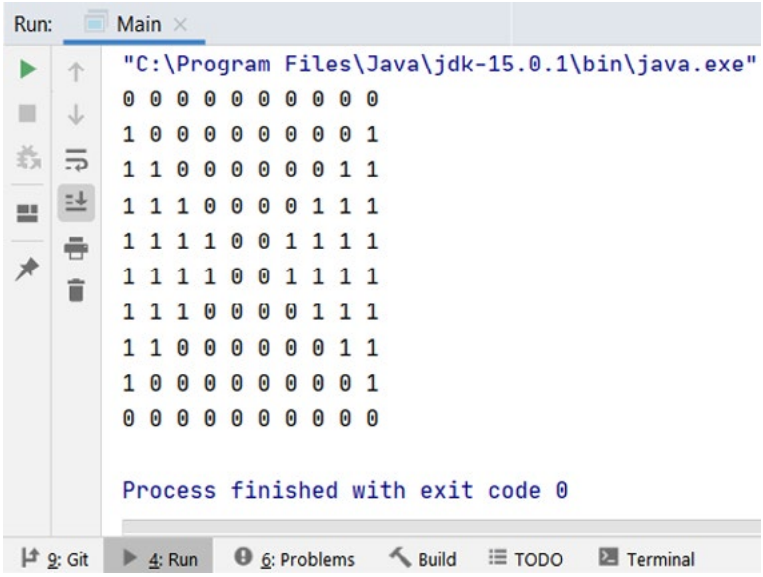

Рис. 63. Результат работы алгоритма «Заборы»

Примечание: простой способ решения данной задачи — последовательно с помощью дополнительного внешнего цикла `for()` накладывать матрицы из нулей и единиц на исходную матрицу 10×10.

Дополнительное задание: обсудить, почему IDE IntelliJ выдаёт предупреждение «Duplicated code fragment» при наведении курсора мыши на объявления массива `int[][] a = new int[10][10]` в методах класса `Main`. Можно ли каким-то образом убрать повторяющиеся объявления из методов? Какие существуют варианты решения этой проблемы?

11. Переписать методы из всех предыдущих задач с матрицами, использовав одномерный массив вместо двумерного: перейти на индексацию вида $i \cdot m+j$.

12. Для массива, заполненного значениями типа double реализовать алгоритм, копирующий элементы с чётными и нечётными индексами в два вспомогательных массива. Вывести результат в консоль.

13.* Создать программу «Склад». Хранилище (класс `Repository`) имеет поле в виде трёхмерной матрицы 20×50×100. Каждый элемент матрицы представляет собой ячейку (класс `Cell`) склада. У ячейки есть свойство — объём, одинаковый для всех ячеек. Некие продукты могут складываться в ячейки и выниматься из них. У продукта (класс `Product`) есть свойства: название и объём. Ячейка считается заполненной, если заполнено более 80% её объёма. Реализовать консольную программу «Склад», которая позволяет: класть некоторый продукт в ячейку и вынимать его из ячейки; выводить содержимое ячейки в консоль; выводить содержимое всего склада в консоль; случайным образом раскладывает продукты по свободным ячейкам. Код взаимодействия с пользователем расположить в методе `main()` класса `TestDrive`; использовать класс `Scanner` для ввода данных.

Выводы: в ходе выполнения лабораторной работы закрепляется материал по массивам (одномерным и двумерным) и управляющим структурам. Дополнительно закрепляются аспекты использования вложенных друг в друга различных управляющих структур: двойных циклов, тройных циклов, ветвлений в циклах.

Контрольные вопросы

1. Чем одномерный массив отличается от двумерного?
2. Можно ли в оператор `if()` вложить `for()`?
3. Можно ли в оператор for() вложить `if()`?
4. Массив, содержащий целые числа, является примитивом или объектом?
5. Что случится, если в цикле `for()` применить команду `break`?
6. Что означает индексация вида $i \cdot m+j$?

Лабораторная работа № 8. Работа со строками

Теоретическая часть

В процессе написания большинства программ необходимо постоянно работать со строковыми данными, т. е. текстовыми выражениями: анализировать строки, генерировать сообщения, формировать ответы, выводить текст в консоль и т. д. Поэтому строковые данные используются практически в любых алгоритмах и программах.

В Java строки определяются как тип данных `String`.

Важно!

Строка в Java является переменной объектного типа. `String` — это специальный класс из стандартного пакета java.lang и представляет собой неизменяемый массив из символов типа char.

Объект типа String можно создать либо инициализировав строковую переменную, либо через конструктор.

```
String name="Вася";
String name2=new("Петя");
```

Важно!

Строковые значения задаются в двойных кавычках, а значения символьных переменных — в одинарных кавычках.

```
String name="Вася";
char c= 'B';
```

Пример работы со строками.

```
import static java.lang.System.*;
public class BasicString {
     public static void main(String[] args) {
          String s=new String("Привет, КУБ");
          String hi="Hello,\t\tCube";
          out.print(s+"\n"+hi+"\n");
          String name="Wasya";
          String welcome="privet";
          String text=name+", "+welcome;
          out.println(text);
     }
}
```

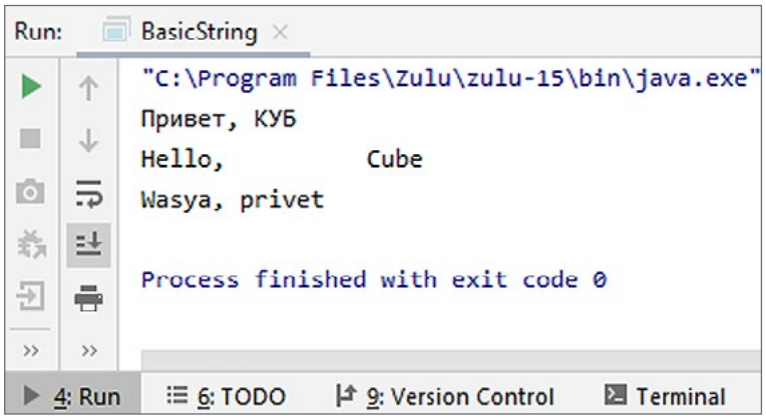

Рис. 64. Базовая работа со строками

Одна из основных операций при работе со строками — конкатенация, т. е. соединение строк. Для операции конкатенации используется оператор +. Строковые значения заключаются в двойные кавычки, а переменные пишутся без кавычек.

```
String s="Привет" +"!"; //соединяются строки
String s2="Привет, "+name; //соединяется строка и переменная
String s3="Привет, "+name+"!"; //соединяется строка, переменная и строка
```

Таким образом можно формировать строковые значения на базе множества переменных и строковых значений.

Основные методы класса `String` для работы со строками

Метод	Описание
`length()`	возвращает длину строки
`charAt(int i)`	возвращает *i*-й символ из строки (поскольку строка — это массив символов, то индексация идёт от 0 до `length()` минус 1)
`substring(int i, int j)`	выбрать подстроку начиная с *i*-го символа до (*j*-1)-го
`split(String x)`	разбить строку на массив подстрок, считая разделителем шаблонную строку *x*
`contains(String s)`	проверка, содержит ли строка символы или строку
`toCharArray()`	преобразует строку в массив символов
`toLowerCase()`	перевод в нижний регистр
`indexOf(String s)`	ищет строку и возвращает индекс её вхождения (если не находит, то возвращает -1)
`indexOf(char c)`	ищет символ и возвращает индекс его вхождения

```
import java.util.Locale;
import static java.lang.System.*;
public class BasicString2 {
    public static void main(String[] args) {
        String s = "IT куб";
```

```
        out.println("Длина строки \"IT куб\"="+s.length());
        char ch=s.charAt(s.length()-1);
        out.println("Последний символ строки \"IT куб\"="+ch);
        out.println(s.substring(3, 6));
        String[] splittedS=s.split(" ");
        for (int i=0; i<splittedS.length; i++) {
            out.println("splittedS["+i+"]="+splittedS[i]);
        }
        out.println(s.contains("IT"));
        char[] chars=s.toCharArray();
        out.println(chars[1]);
        s.toLowerCase(Locale.ROOT);
        out.println(s);
        out.println("Индекс символа T="+s.indexOf('T'));
      out.println("Индекс начала строки IT="+s.indexOf("IT"));
    }
}
```

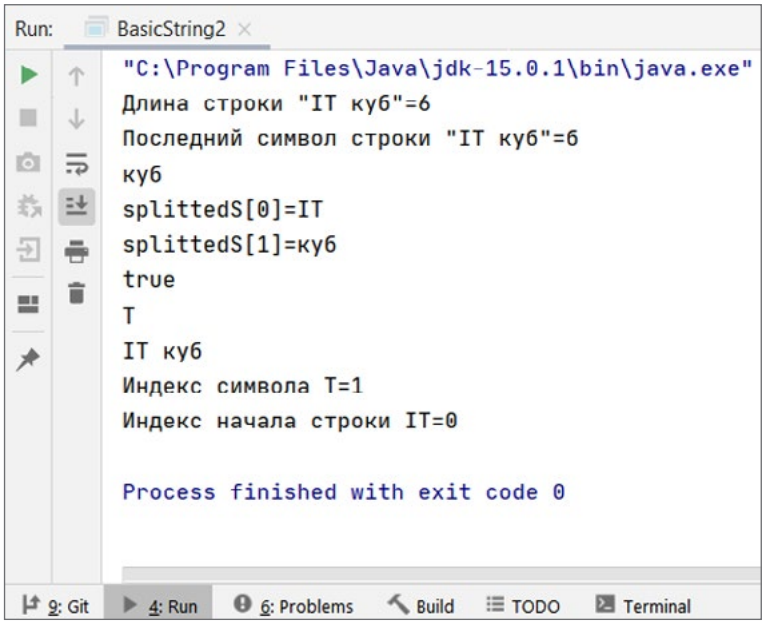

Рис. 65. Результат работы методов класса `String`

Важно!

В примере выше видно, что для вывода символа двойных кавычек необходимо перед знаком кавычек поставить обратный слэш (\), иначе двойная кавычка будет восприниматься как завершение или начало строковых данных.

Также можно обратить внимание на ещё одно полезное применение конкатенации строк в следующей конструкции.

```
out.println("splittedS["+i+"]="+splittedS[i]);
```

Подобные конструкции применяются для вывода меняющегося в цикле значения индекса *i* с помощью строкового выражения.

Элемент типа String представляет из себя неизменяемую строку, поэтому напрямую редактировать его нельзя. Но можно создать новый экземпляр класса.

```
String s="Вася";
s=new String("Петя");
```

Важно!

Даже если в редакторе IntelliJ присвоить строковой переменной новое значение, на самом деле произойдёт создание нового экземпляра класса `String`, для удобства скрытого от глаз пользователя.

В случае множественных манипуляций со строкой может быть удобным использовать специальный класс `StringBuilder`, который создаётся либо с пустым конструктором, либо сразу инициализируется по строковой переменной.

```
StringBuilder stringBuilder1 = new StringBuilder();
StringBuilder stringBuilder2 = new StringBuilder(s);
```

Полезными могут оказаться такие методы класса `StringBuilder`, как `replace()` — замена символов в строке, `insert()` — вставка символов, `delete()` — удаление символов, `reverse()` — обращение строки.

```
import static java.lang.System.*;
public class BasicString3 {
    public static void main(String[] args) {
        String s = "IT куб";
        StringBuilder stringBuilder = new StringBuilder(s);
        out.println(stringBuilder.reverse());
        out.println(stringBuilder.reverse());
        out.println(stringBuilder.replace(0,2,"it"));
        out.println(stringBuilder.insert(6,"ъ"));
        out.println(stringBuilder.delete(0,3));
    }
}
```

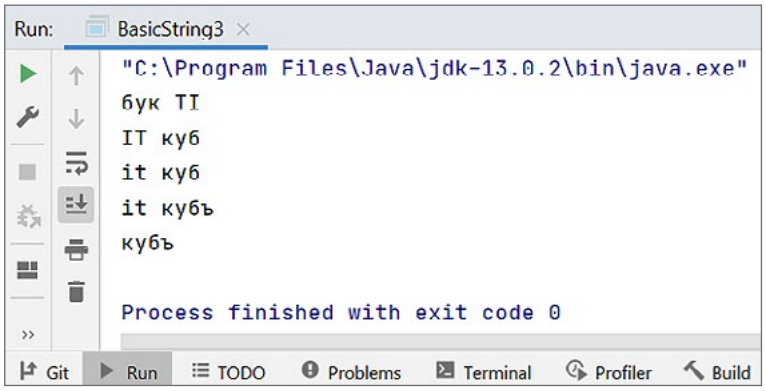

Рис. 66. Результат работы методов класса `StringBuilder`

Практическая часть

Цель работы: ознакомление с базовыми принципами работы со строковыми данными.

Ход лабораторной работы

1. Запустить среду IntelliJ. При необходимости создать новый проект и класс с методом `main()`. Выполнить все примеры из теоретической части.
2. В предложении «На улице стоял рояль с нотами» убрать все пробелы.
3. Предложение «По вечерам все музыканты города собирались на набережной» преобразовать в массив строк по словам, считая пробел разделителем слов.
4. Найти сумму чисел, встречающихся в строке. (Можно использовать метод `Integer.parseInt()` для преобразования строки в число).
5. Ввести две строки через консоль. Найти самую короткую и самую длинную строки.
6. Из строкового выражения «Добрый день!» составить две строки: первая должна содержать нечётные символы по индексу («орйдн!») исходного выражения, вторая — чётные (Дбы ень»). Написать метод, который принимает в качестве параметров первую и вторую строку, а в качестве результата возвращает исходную строку («Добрый день!»).
7. Написать метод, разворачивающий символы строки справа налево.
8. Написать метод, преобразующий символы строкового выражения на кириллице в символы на латинице.
9. В строке «IT-куб преобразуется в IT-инкубатор» найти индекс второго вхождения буквы «а» и индекс второго вхождения строки «IT».
   Подсказка:с помощью метода `indexOf()` можно найти не только первое вхождение символа или строки в исходную строку, но и любое *n*-ное вхождение символа. Существует несколько вариантов это сделать. Можно неоднократно использовать метод `indexOf()`, например, в цикле и в комбинации с методом `substring()`, постепенно уменьшая исходную строку, вырезая из неё части слева. Также можно воспользоваться методом `charAt()` в цикле, проходя по массиву символов строки.

Выводы: в ходе выполнения лабораторной работы произошло ознакомление с базовыми принципами работы со строковыми данными.

Контрольные вопросы

1. Как объявляются и инициализируются строки в языке Java?
2. Какие методы есть у класса `String`?
3. Что делает метод `reverse()`?
4. Как определяется длина строки и как выполняется сложение строк в языке Java?
5. Зачем нужно работать со строковыми данными?
6. Чем символ отличается от строки?

Лабораторная работа № 9.1. Списки

Теоретическая часть

Для хранения и обработки большого количества данных удобно использовать массивы, в особенности для операций над группами объектов. Однако у массивов в Java есть недостатки. В отличие от некоторых других языков, где массивы представляют собой динамические структуры, в Java невозможно изменить размер массива, так как при его создании указывается количество элементов и резервируется необходимая для них память. В Java нет встроенных методов для работы с элементами массива: поиска, итерации, до-

бавления, удаления и т. п. Работа с массивами происходит строго с учётом индексации. Для расширения возможнотей работы с массивами приходится создавать свой класс и описывать в ней свою структуру данных, содержащую массивы, и реализовывать необходимый функционал.

Поэтому наряду с массивами в языке Java существуют и другие структуры данных. В данной лабораторной работе мы рассмотрим динамический список — класс `ArrayList`. С помощью его методов возможно изменение размера, удаление, добавление новых элементов.

Важно!

Индексация элементов списка также начинается с нуля, как и у массивов.
В основе реализации `ArrayList` лежит обычный массив, но когда выделенная память заполняется, выделяется новая память, в 1,5 раза больше предыдущей, после чего в новый массив копируются элементы старого и тот удаляется.

Типовое объявление и создание динамического списка `ArrayList` выглядит следующим образом.

```
ArrayList<String> myStrings=new ArrayList<>();
```

Здесь `<String>` означает параметризованный тип. То есть в данном списке с именем `myStrings` могут храниться только объекты типа `String` и никакие другие. Это очень удобно для управления содержимым списков и с точки зрения безопасности кода.

Здесь список сразу создан и объявлен в одной инструкции. При необходимости процесс можно разбить на два действия.

```
ArrayList<String> myStrings; //объявление myStrings
myStrings = new ArrayList<>(); //создание объекта
```

В этом случае до создания объекта нельзя работать с `myStrings`, иначе Java выдаст сообщение об ошибке и программа досрочно завершится.

Если указывается точное количество элементов списка, то память резервируется под заданное количество элементов списка. Создаётся такой список следующим образом.

```
ArrayList<String> myStrings=new ArrayList<>(10);
```

Возможно сразу создать проинициализированный список.

```
ArrayList<String> itCubeCourses = new
ArrayList<String>(Arrays.asList(«Java», «Mobile», «Robot»,
«Python», «Scratch»));
```

В качестве параметризованного типа при создании динамического списка нельзя указывать примитивные типы данных, потому что они не являются классами. Для решения этой проблемы можно использовать так называемые классы-обёртки над примитивными типами. Классы-обёртки существуют для того, чтобы можно было реализовать объектно-ориентированный подход для примитивов.

```
ArrayList<Integer> nums = new ArrayList<>();
ArrayList<Double> dnums = new ArrayList<>();
ArrayList<Boolean> oks = new ArrayList<>();
```

Можно использовать собственные классы в качестве параметров динамического списка.

```
ArrayList<Man> men=new ArrayList<>();
```

В этом случае список будет содержать объекты класса `Man`. Соответственно с любым из элементов списка можно будет работать как с объектом класса `Man`: вызывать методы, обращаться к полям и т. д.

Основные методы, необходимые для работы с динамическими списками

`add(element)`	добавление элемента в конец списка
`add(index, element)`	добавление элемента с учётом позиции
`addAll(index, collection)`	добавление коллекции элементов
`set(index, element)`	изменение значения элемента
`get(index)`	получение элемента по индексу
`indexOf(element)`	получение индекса элемента
`isEmpty()`	проверка, является ли список пустым
`remove(index)`	удаление элемента по индексу
`contains(element)`	проверка на наличие элемента
`size()`	получение количества элементов списка
`clear()`	очистка списка

Примеры работы со списками.

```
package com.itcube.ds;
import static java.lang.System.out;
import java.util.ArrayList;
import java.util.Arrays;
public class SimpleArrayList {
    public static void main(String[] args) {
        ArrayList<String> itCubeCourses =
            new ArrayList<String>(Arrays.asList("Java",
"Mobile","Robot","Python","Scratch","Pascal"));
        ArrayList<String> names =
            new ArrayList<String>(10);
        ArrayList<Integer> numbers =
            new ArrayList<Integer>(Arrays.asList(1, 2, 3, 5,
6));
        ArrayList<Integer> numbers2 =
           new ArrayList<Integer>(Arrays.asList(7, 8, 9, 10));
        out.println(names.contains("Robot")); //проверка, есть
ли строка Robot в списке
        out.println(itCubeCourses.contains("Robot"));
        //проверка, есть ли строка Robot в списке
        out.println(names.isEmpty());
        //является ли список пустым
        out.println(numbers);
        //вывести в консоль все элементы списка
        numbers.add(3, 4);
```

```
            //добавить на 3-й индекс число 4, сместив элементы
вправо
            out.println(numbers);
            numbers.addAll(6, numbers2);
            //добавить список numbers2 в numbers
            //начиная с 6-го индекса
            System.out.println(numbers);
            out.println("numbers.get(0)="+numbers.get(0));
            //вывести элемент с 0-м индексом
            out.println("numbers.get(size-1)="+numbers.get(numbers.
size()-1));
            //вывести последний элемент
  out.println("itCubeCoreses.indexOf(Robot)="+itCubeCourses.
indexOf("Robot"));
            //получить индекс элемента Robot
            for(int i=0; i<itCubeCourses.size();i++){
                out.print(itCubeCourses.get(i)+"\t");
            }//вывести в консоль значения элементов itCubeCourses
            out.println();
            out.println(itCubeCourses.get(1)=="Pascal");
            //проверка, является ли "Pascal" первым в списке
itCubeCourses
        }
    }
```

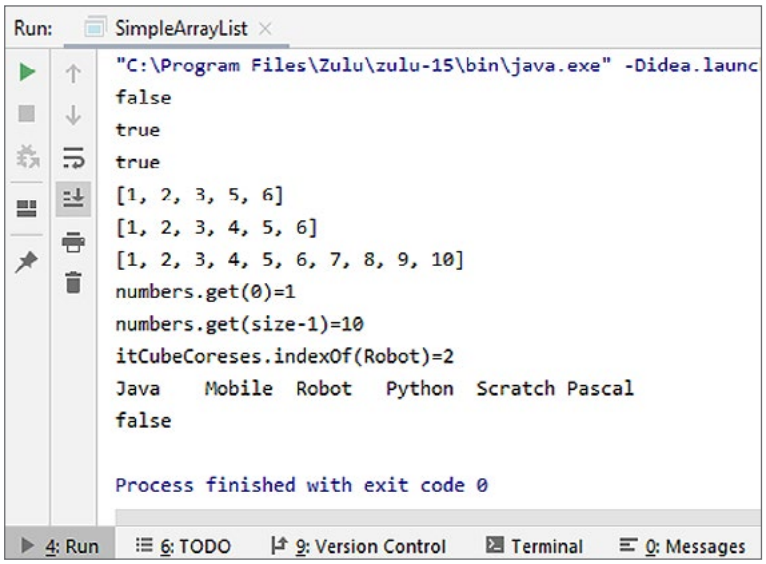

Рис. 67. Результат работы методов класса `ArrayList`

Для перебора элементов списка можно использовать цикл `for()`.

```
for(int i=0; i<numbers.size(); i++){
    out.println(numbers.get(i));
}
```

Также существует способ прохождения по элементам списка как по коллекции, не инициализируя счётчики и не определяя условия работы цикла. В этом случае отпадает

необходимость пользоваться индексами, которые обеспечиваются счётчиком в обычном цикле `for()`. Циклы с итерацией по коллекции по-другому называются foreach-циклы.

Синтаксис таких циклов в Java следующий.

```
for(Переменная типа параметра списка: имя списка){
    операции с переменной
}
```

Пример.

```
import static java.lang.System.out;
import java.util.ArrayList;
import java.util.Arrays;
public class SimpleArrayList {
    public static void main(String[] args) {
        ArrayList<String> itCubeCourses=
                new ArrayList<String>(Arrays.asList("Java",
"Mobile","Robot","Python","Scratch","Pascal" ));
        for (String s : itCubeCourses) {
            out.print(s+"\t");
        }
        out.println();
        itCubeCourses.remove("Pascal");
        out.println(itCubeCourses.contains("Pascal"));
    }
}
```

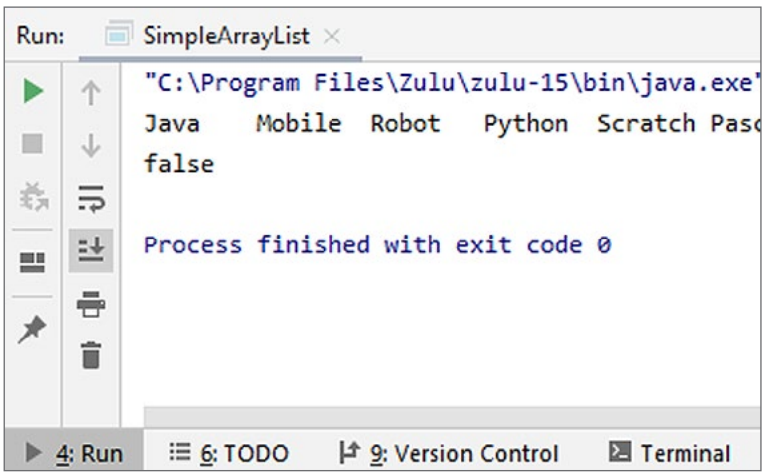

Рис. 68. Результат работы со списком с помощью foreach-цикла

Циклы типа foreach, по сути, являются итераторами и очень удобны тем, что, используя их, пользователь избавляется от необходимости инициализировать индексы, условия, шаги, и без лишних настроек перебирает элементы списка. Однако основное предназначение foreach — перебирать значения списка, а не удалять или добавлять новые. Также при работе с динамическим списком можно использовать встроенный итератор, возвращаемый методом `iterator()`. Кстати, именно на нём основан цикл foreach.

```
    public static void main(String[] args) {
        ArrayList<String> itCubeCourses=
                new ArrayList<String>(Arrays.asList("Java","Mobile",
"Robot","Python","Scratch","Pascal","Turbo C"));
        Iterator<String> iterator = itCubeCourses.iterator();
        while (iterator.hasNext()) {
            String s=iterator.next();
            out.println(s);
        }
    }
```

Циклом foreach можно перебирать не только элементы списка, но и элементы обычного массива.

```
int [] numbers = {41, 5, 5, 5, 2, 7};
for (int number: numbers) {
    out.println(number);
}
```

Практическая часть

Цель работы: получить навыки работы с динамическими списками в Java.

Ход лабораторной работы

1. Запустить среду IntelliJ. При необходимости создать новый проект и класс с методом `main()`.
2. Реализовать все примеры из теоретической части. Выполнить и проанализировать результаты.
3. Создать список типа `String`.
4. Заполнить его значениями имён учеников группы.
5. Вывести содержимое списка в консоль, используя обычный `for()`, foreach-цикл, вывод списка с помощью `out.println()`.
6. Вывести в консоль значения чётных элементов списка.
7. Вывести в консоль размер списка.
8. Удалить нечётные элементы списка и вывести результат в консоль.
9. Создать второй список типа `String` и заполнить его произвольными именами. Объединить оба списка и вывести результат в консоль.
10. Написать метод для сортировки списка.
11. Создать список, заполненный элементами от 0 до 10, затем поменять число 7 на число 42.
12. Создать список типа `int` с длиной не менее 10 и удалить половину наименьших по значению элементов списка.
13. Создать список типа `String` длины 10 с изначально инициализированными значениями. Удалить дублирующиеся значения.
14. Создать `ArrayList<Man>` и заполнить его пятью произвольными значениями. Класс Man взять из лабораторной работы № 5.1 «Классы». Вывести в консоль значения переменной `name` всех объектов с помощью итератора.
15. Самостоятельно изучить, что представляют из себя структуры вида `ArrayList<ArrayList<String>>`.

Выводы: в ходе выполнения лабораторной работы получены навыки по работе с динамическими списками в Java.

Контрольные вопросы

1. Чем отличается список от массива?
2. Почему у списка есть несколько методов для поиска элемента?
3. С какими типами данных может работать список?
4. Зачем у списка есть конструктор с указанием количества элементов, под которые нужно резервировать место, если он динамически расширяется?
5. Может ли список хранить объекты любых типов?

Лабораторная работа № 9.2. Двумерные списки

Теоретическая часть

Полезным функционалом при работе со списками обладает специальный класс `Collections`.

Методы класса Collections

Методы	Описание
`sort(список)`	сортировка списка
`frequency(список, элемент)`	нахождение частоты элемента
`fill(список, элемент)`	заполнение списка одинаковыми элементами

Пример использования методов класса Collections.

```
public static void main(String[] args) {
     ArrayList<String> itCubeCourses=
          new ArrayList<String>(Arrays.asList("Java", "Pascal",
"Pascal","Python","Scratch","Pascal","Turbo C"));
     Collections.sort(itCubeCourses);
     itCubeCourses.forEach(out::println);
     int number = Collections.frequency(itCubeCourses, "Pascal");
     out.println(number);
     Collections.fill(itCubeCourses, new String("test"));
     itCubeCourses.forEach(out::println);
}
```

В этом примере использован альтернативный упрощённый вывод данных списка в консоль.

```
itCubeCourses.forEach(out::println);
```

За счёт этой конструкции сокращается количество строк кода по сравнению с использованием обычного цикла `for()`. На данном этапе её следует принять как данность, без синтаксического разбора.

Аналогом двумерного массива для динамических списков может служить двойной список, т. е. список, содержащий список списков.

```
ArrayList<ArrayList<String>> aaList = new
ArrayList<ArrayList<String>>();
```

Подобный двумерный список состоит из списков `ArrayList<String>`, которые, в свою очередь, хранят элементы типа `String`. Поэтому для работы с таким списком сначала нужно создать его внутренние списки, а уже затем работать с их содержимым.

```
public static void main(String[] args) {
    ArrayList<ArrayList<String>> arrayLists = new ArrayList<>();
    arrayLists.add(new ArrayList<String>());
    arrayLists.add(new ArrayList<String>());
    arrayLists.add(new ArrayList<String>());
    arrayLists.get(0).add("Текст 1-го списка");
    arrayLists.get(1).add("Текст 2-го списка");
    arrayLists.get(2).add("Текст 3-го списка");
    out.println(arrayLists.get(0).get(0));
    out.println(arrayLists.get(1).get(0));
    out.println(arrayLists.get(2).get(0));
}
```

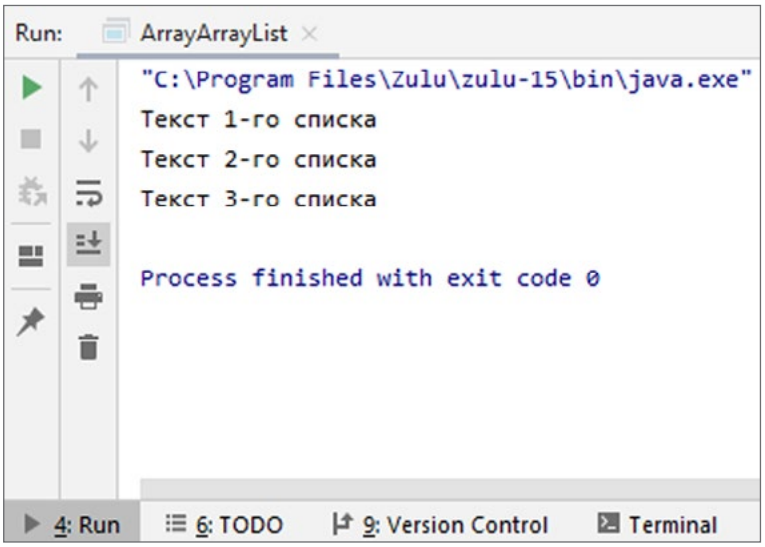

Рис. 69. Результат работы с двумерным списком

Для удобства можно создать ссылку на любой из внутренних списков и работать с ним отдельно.

```
ArrayList<String> firstList=llist.get(0);
firstList.add("Новый текст в 1-й список");
```

Как известно из предыдущих лабораторных работ, списки в Java могут использовать параметризованные типы, т. е. можно указывать, объекты какого типа должны храниться в списке. При этом типом объекта может быть не только String, но и любые другие классы. Например, имеется класс `Car`.

```
public class Car {
    private boolean electric;
    private boolean oil;
    private boolean autoPilot;
    private String name;
    public Car(String name, boolean electric, boolean oil,
boolean autoPilot) {
```

```
        this.name=name;
        this.electric = electric;
        this.oil = oil;
        this.autoPilot = autoPilot;
    }
    public boolean isElectric() {
        return electric;
    }
    public boolean isOil() {
        return oil;
    }
    public boolean isAutoPilot() {
        return autoPilot;
    }
    public String getName() {
        return name;
    }
    @Override
    public String toString() {
        return "Car{" +
            "electric=" + electric +
            ", oil=" + oil + ",
            autoPilot=" + autoPilot +
            ", name=\'" + name + '\" +
            '}';
    }
}
```

Класс `Car` можно использовать для параметризации списка.

```
public static void main(String[] args) {
    ArrayList<Car> myCars = new ArrayList<>();
    myCars.add(new Car("КАМАЗ",false,true,false));
    myCars.add(new Car("УАЗ",false,true,false));
    Car car1=myCars.get(0);
    Car car2=myCars.get(1);
    out.println(car1);
    out.println(car2);
}
```

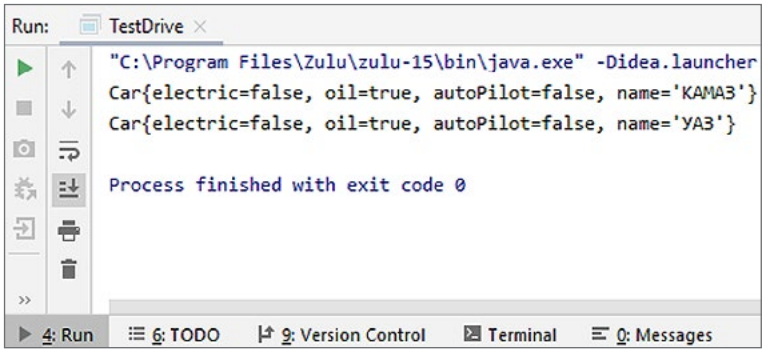

Рис. 70. Результат работы со списком, параметризованным типом `Car`

Здесь класс `Car` в некотором смысле представляет собой не только класс, но и структуру данных, т. к. он организован так, что хранит много полей с информацией о свойствах машины. С этой точки зрения класс `Car` можно рассматривать как некую переменную, представляющую из себя структуру, в которой хранится набор значений.

В классе `Car` представлен метод `toString()`, который имеется во всех классах по умолчанию. Если его не поменять (не переопределить), тогда если вывести в консоль сам объект с помощью `out.println(car1)`, то результатом будет идентификатор (хэш-код) объекта следующего вида: `com.itcube.cs.Car@378bf509`.

Если метод `toString()` переопределить, то можно задать структуру и вид строки, которую этот метод возвращает:

```
@Override //директива переопределения метода
public String toString() {
     return "Car{" +
          "electric=" + electric +
          ", oil=" + oil +
          ", autoPilot=" + autoPilot + ", name=\" + name +
          '\' ' + '}';
}
```

После такой манипуляции результатом инструкции `out.println(car1)` будет вывод в консоль информации в следующем виде: «Car{electric=false, oil=true, autoPilot=false, name='КАМАЗ'}».

Самое интересное, что такую форму метода сама среда может автоматически сгенерировать на основе анализа полей класса `Car`. Для этого необходимо проделать следующий действия.

1. Установить курсор мыши после любого метода класса `Car`.
2. Воспользоваться сочетанием клавиш *Alt+Ins*. Появится контекстное меню.

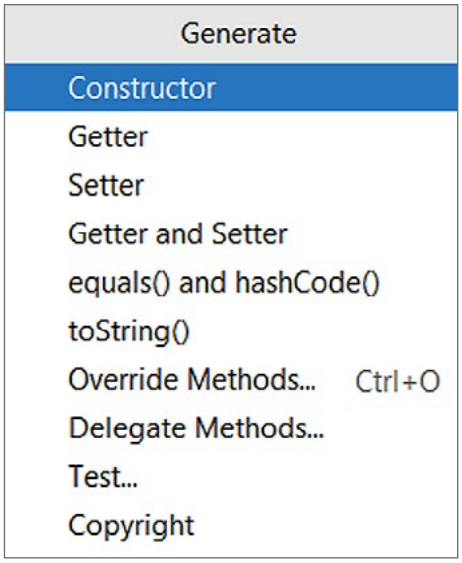

Рис. 71. Контекстное меню, вызванное с помощью *Alt+Ins*

3. В появившемся контекстном окне щёлкнуть мышью по *toString()*. Появится диалоговое окно.

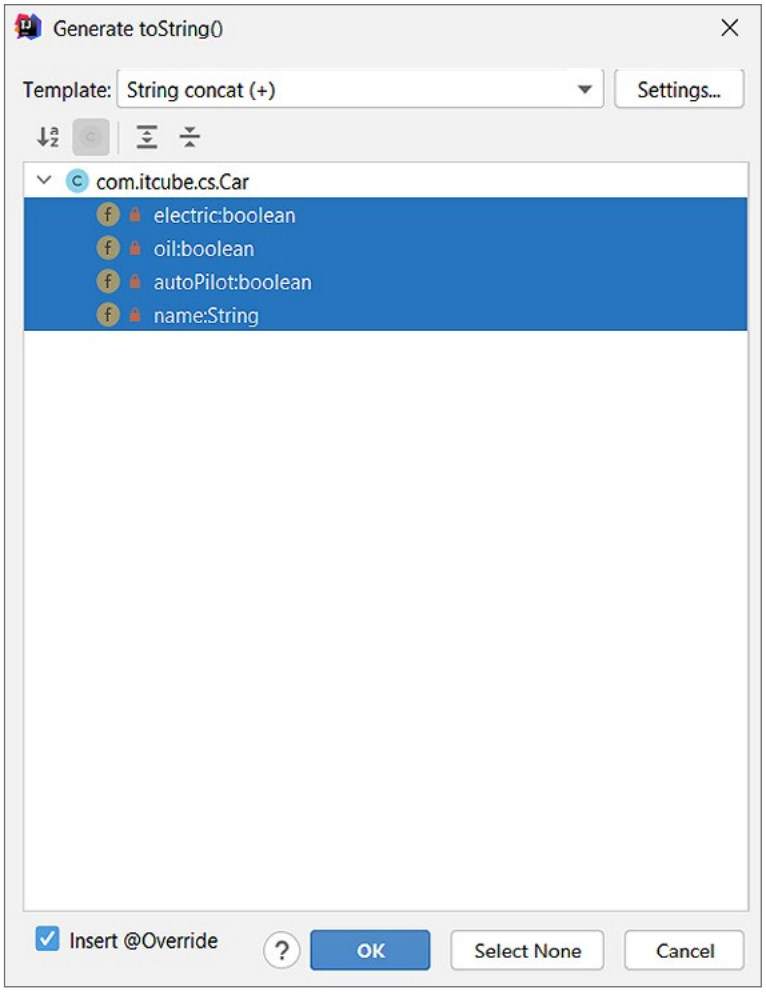

Рис. 72. Диалоговое окно переопределения метода `toString()`

4. В диалоговом окне оставить всё как есть или выбрать переменные, которые будут выводиться в методе, и нажать *OK*.

Переопределённый метод появится в том месте, где был установлен курсор мыши.

Таким же образом через сочетание *Alt+Ins* можно генерировать не только метод `toString()`, но и геттеры, сеттеры, конструкторы. Всего за несколько кликов мыши среда IntelliJ сама создаст нужную конструкцию.

Практическая часть

Цель работы: закрепить навыки работы с динамическими списками в Java.

Ход лабораторной работы

1. Запустить среду IntelliJ. При необходимости создать новый проект и класс с методом `main()`.
2. Реализовать все примеры из теоретической части.

Дополнить класс `Car` следующими методами (каждый метод генерирует экземпляр класса `Car` с определёнными характеристиками, например: метод `hybridAutoPilotCar()` генерирует машину, которая использует и электричество, и бензин и обладает функцией автопилота).

```
public static Car hybridAutoPilotCar(String name) {
    return new Car(name,true,true,true);
}
```

```
public static Car hybridManCar(String name) {
     return new Car(name,true,true,false);
}
public static Car electroAutoPilotCar(String name) {
     return new Car(name,true,false,true);
}
public static Car electroManCar(String name) {
     return new Car(name,true,false,false);
}
public static Car oilAutoPilotCar(String name) {
     return new Car(name,false,true,true);
}
public static Car oilManCar(String name) {
     return new Car(name,false,true,false);
}
public static Car rikshaCar(String name) {
     return new Car(name,false,false,false);
}
```

4. Создать список, параметризованный типом `Car`, и проинициализировать его значениями, воспользовавшись методом `asList()` специального класса `Arrays`.

```
//список машин
     ArrayList<Car> cars = new ArrayList<Car>(Arrays.asList(
          Car.oilManCar("Lada"),
          Car.electroAutoPilotCar("Tesla1"),
          Car.hybridAutoPilotCar("BMW"),
          Car.electroManCar("Tesla2"),
          Car.oilManCar("Moskvich"),
          Car.rikshaCar("Riksha") ));
```

5. В методе `main()` реализовать код, который выводит в консоль только те машины из списка, для которых соблюдается условие:

а) машина работает на автопилоте;

б) машина работает на автопилоте и на бензиновом двигателе.

Примерный код для реализации пунктов может быть следующим:

а)

```
for (Car c : cars) {
     if (c.isAutoPilot()) {
          out.println(c.getName()+" может работать на
автопилоте");
     }
}
```

б)

```
for (Car c : cars) {
     if (c.isAutoPilot()&&c.isOil()) {
          out.println(c.getName()+" автопилот И бензин");
     }
}
```

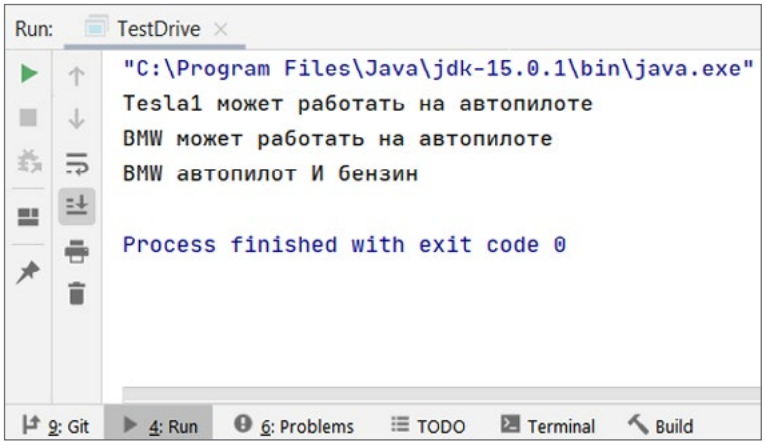

Рис. 73. Результат выполнения задания для пунктов *а* и *б*

6. Аналогично пункту 5 реализовать следующие условия:

а) машина не работает на автопилоте;

б) машина работает только на электричестве;

в) машина работает либо на бензине и не работает на автопилоте, либо работает только на автопилоте, но не работает на бензине. *Подсказка*: можно использовать исключающее ИЛИ;

г) машина работает на электричестве и автопилоте;

д) машина не работает на электричестве и не работает на бензине;

е) машина не работает на автопилоте и является электрической.

7. Создать динамический список `ArrayList<String> secondNames` с фамилиями учеников текущего курса «IT-Куб». Фамилии можно добавить через метод `add()`, либо сразу проинициализировать с помощью метода `Arrays.asList()`
8. Отсортировать список с помощью метода `Collections.Sort()` и вывести на консоль тремя способами:

а) через обычный цикл `for()`;

б) через цикл типа foreach;

в) через упрощённый вариант `secondNames.forEach(out::println)`.

9. Найти в списке `secondNames` количество элементов со значением «Иванов».
10. Из списка `secondNames` создать новый список `uniqueNames`, в котором содержались бы только уникальные фамилии, без дублирования.
11. Создать двумерный массив и вывести его в консоль.

```
int[][] matrix={ {1,1,1}, {2,2,2}, {3,3,3}}; }
```

12. Создать пустой двумерный список `ArrayList<ArrayList<Integer>> nums` и заполнить его тремя пустыми одномерными списками `ArrayList<Integer>`. В двойном цикле `for()` заполнить список значениями массива. Вывести значения nums в консоль. Результат не должен отличаться от вывода значений массива `matrix`.

Лабораторная работа № 10. Отладка кода

Теоретическая часть

Отладка кода (*debug*, *debugging*) — это необходимый и чрезвычайно важный этап разработки программ. С помощью отладки кода можно найти ошибки, которые трудно заметить при простом чтении и изучении кода.

Важно!

Отладка помогает найти те ошибки, которые связаны с неправильной реализацией алгоритмов и не отслеживаются средой на этапе компиляции.

В режиме отладки код запускается и выполняется пошагово, т. е. происходит построчный проход по коду в ожидании команды пользователя о переходе к каждой следующей команде. На каждом шаге отладки можно увидеть значения всех переменных, полей объектов и т. д.

С помощью отладки кода программисты могут пошагово наблюдать за порядком выполнения программы. Учащиеся могут следовать за потоком управления программы и лучше понимать, как работают алгоритмы, как поток управления в зависимости от условий переходит на те или иные блоки кода.

Для обозначения важных моментов, на которых нужно остановиться при отладке, используются точки остановки (*breakpoint*). При запуске процесс отладки начинается с места первой точки остановки.

В IntelliJ существует два основных режима прохода по коду:

1) «step over» (клавиша *F8*) — обычный проход по строкам без захода в реализацию метода;

2) «step into» (клавиша *F7*) — проход по строке с заходом в метод.

Отличие захода внутрь от обычного прохода в том, что в этом случае отладчик проходит по коду максимально глубоко, насколько возможно, следуя за потоком управления. Например, если в строке 17 есть вызов метода `isOk(b)`, то при «step over» отладчик на одном шаге выполнит вызов метода и перейдёт на строку 18, а в случае «step into» отладчик перейдёт в тот класс, где находится метод `isOk()`, и начнёт пошагово выполнять инструкции метода. По завершении работы метода отладчик вернётся на строку 17 и перейдёт на строку 18. Существует переход внутрь методов по выбору из предложенных с помощью сочетания клавиш *Shift+F7*.

На самом деле режимов прохода по строкам намного больше: переход к курсору (когда отладчик можно сразу перевести на строку, где установлен курсор мыши), форсированный проход в инструкции строки и т. д. В рамках данного курса вполне достаточно рассмотреть два основных режима отладки.

Рассмотрим отладку на следующем примере нахождения суммы элементов массива.

```
import static java.lang.System.out;
public class Simple {
    public static void main(String[] args) {
        int[] data = {100, 200, 300, 500, 2000, 20000, 100000};
         int summa=0;
         for (int i=0; i<data.length-1; i++) {
              summa += data[i];
         }
         out.println("Сумма="+summa);
    }
}
```

Сумма элементов должна быть равна 123 100, в консоль же выводится 23 100. При этом программа выполняется и не возникает ошибок компиляции. В этом случае необходимо воспользоваться отладчиком.

Необходимо установить точку остановки нажатием кнопки мыши справа от номера нужной строки, в результате чего слева появится знак в виде красного кружка. Сама строка при этом подсветится розовым цветом.

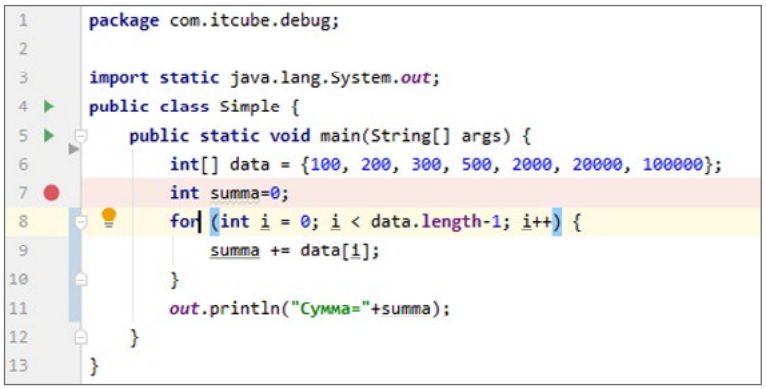

Рис. 74. Установка точки остановки

Далее необходимо в верхнем меню выбрать пункт *Run — Debug* и в появившемся контекстном меню выбрать текущий класс.

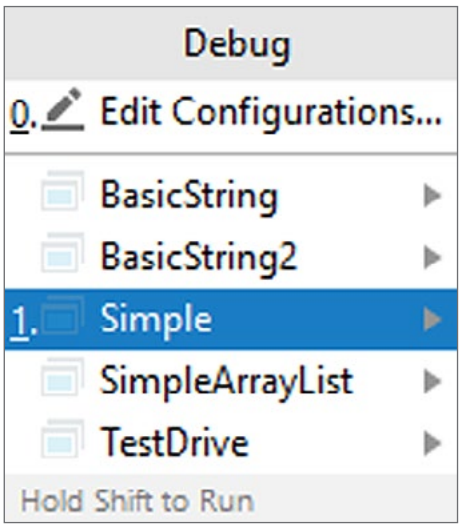

Рис. 75. Выбор класса для отладки

После выбора класса экран IDE переходит в режим отладки.

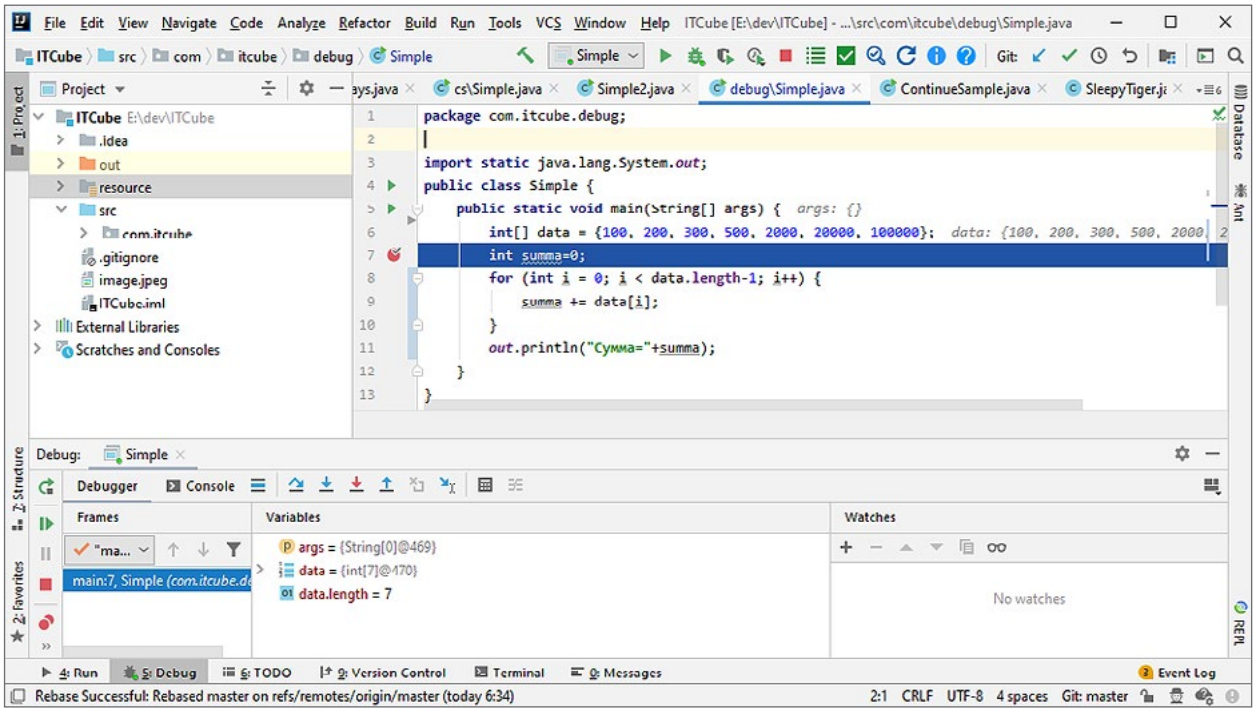

Рис. 76. Режим отладки

В режиме отладки вам будут показаны текущие значения переменных. Их можно отслеживать в окне *Variables*, где они представлены в виде списка со своими текущими значениями.

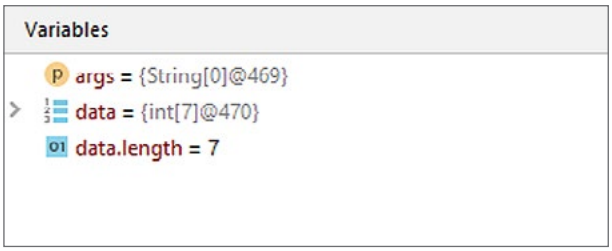

Рис. 77. Окно Variables

В окне *Variables* можно посмотреть все значения массивов и списков, раскрывая их нажатием на знак >.

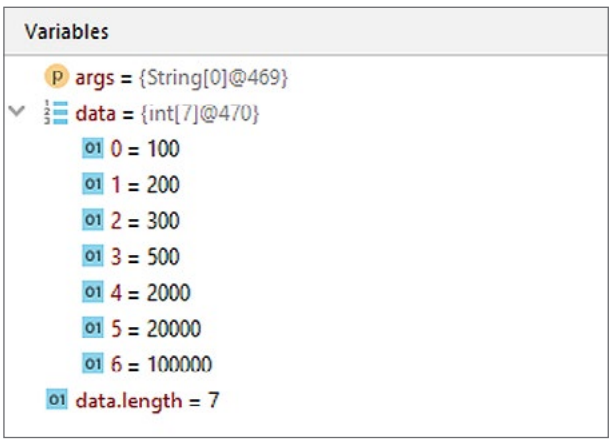

Рис. 78. Просмотр всех значений массива

Кроме того, при наведении курсора мыши на любой объект в коде появится его идентификатор и возможность детального просмотра его содержимого в контекстном окне.

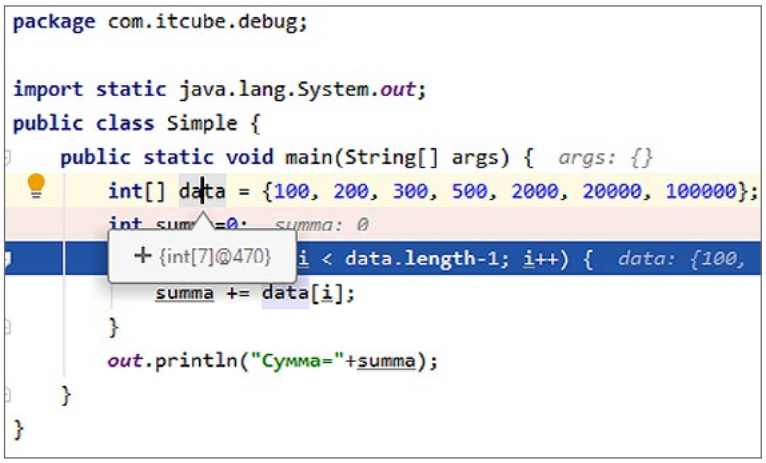

Рис. 79. Просмотр текущего содержимого объекта в окне кода

При нажатии на + показывается содержимое массива. При работе с объектами классов при таком подходе показывается содержимое объекта — текущие переменные, которые он хранит.

Для перехода на следующий шаг нужно нажать *F8*. Тогда синяя строка, отображающая текущее положение отладчика, перейдет на следующую строку и зайдёт в цикл.

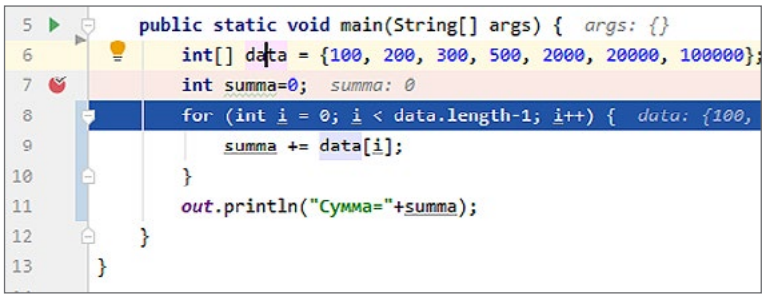

Рис. 80. Переход отладчика по строкам

При следующих нажатиях *F8* отладчик будет ходить по циклу, передвигаясь по строкам 8 и 9 до тех пор, пока цикл не завершится.

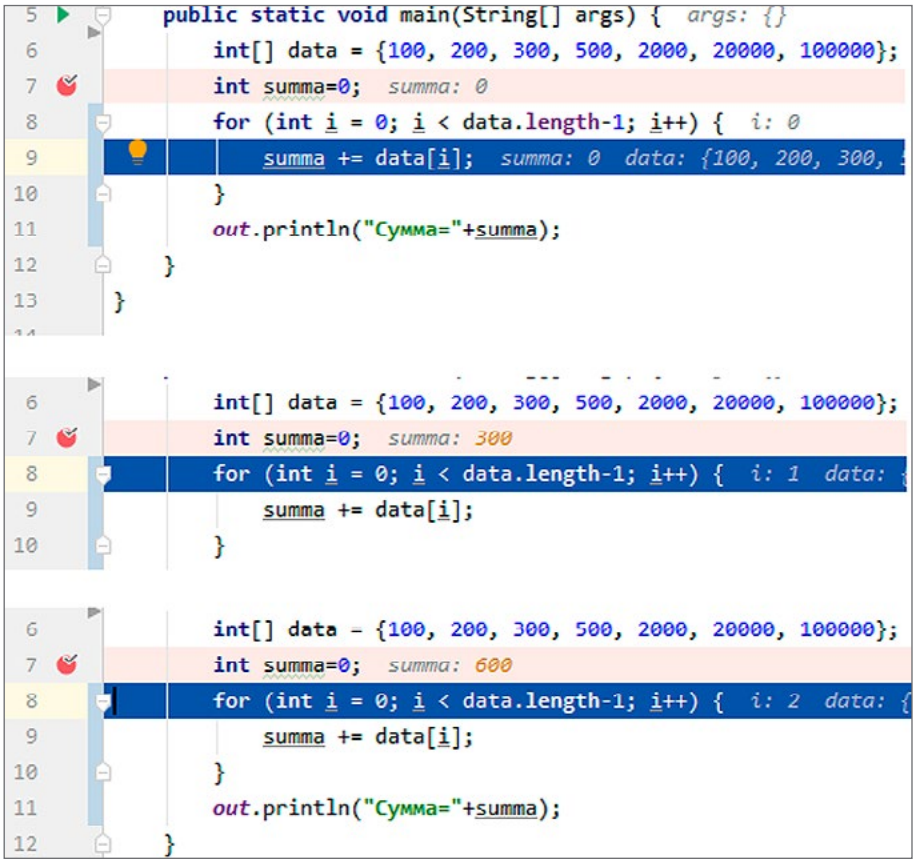

Рис. 81. Проход отладчика по циклу

При этом в строке 7 высвечивается значение переменной *summa* для каждого шага. Также можно в любой момент подвести курсор мыши к переменной и посмотреть её текущее значение.

Для просмотра текущего значения счётчика цикла также можно подвести к нему курсор мыши либо увидеть его значение, высвечиваемое в строке 8.

Проходя таким образом по циклу, можно заметить, что он завершается при *i*=5, хотя в массиве *data* содержится 6 значений, из чего мы и делаем вывод об ошибке в написании условия цикла `i<data.length-1`. Из-за ошибочного присутствия в условии «-1» последний элемент массива не суммируется.

Чтобы досрочно остановить отладку, можно либо выбрать пункт меню *Run — Stop*, либо нажать *Ctrl+F2*, либо нажать на значок с красным квадратом на панели отладки. Также в окне *Frames* можно удалить контекст вызова функции, нажав на верхний значок со стрелкой. Это позволяет откатиться к предыдущей точке остановки и провести отладку цикла заново.

Практическая часть

Цель работы: освоить работу с отладчиком кода в среде IntelliJ.

Ход лабораторной работы

1. Запустить среду IntelliJ. При необходимости создать новый проект и класс с методом `main()`.
2. Реализовать пример из теоретической части.
3. В программе для нахождения факториала найти и исправить ошибки с помощью отладки, используя одну точку остановки.

```
public static void main(String[] args) {
     int number=10;
     int result=0;
     for (int i=0; i <number; i++) {
          result=result*i;
     }
     System.out.println(result);
}
```

4. Реализовать в методе `main()` код поиска чётных чисел. Использовать отладку и найти ошибки.

```
int number=10;
for(int i=0;i<=number;i+=2){
     if(i%2==0) {
          System.out.println(«Чётное число»);
          i++;
     }
}
```

5. Найти в коде метода сортировки «пузырьком» ошибку, используя отладку IntelliJ. В данном варианте код выполняется бесконечно.

```
int[] numbers={41, 5, 5, 5, 2, 7};
boolean isSorted=false;
int stakan;
while(!isSorted) {
     isSorted=true;
     for (int i=0; i<numbers.length-1; i++) {
          if(numbers[i]>=numbers[i+1]){
               isSorted=false;
               stakan=numbers[i];
               numbers[i]=numbers[i+1];
               numbers[i+1]=stakan;
          }
     }
     out.println("while ...?");
}
for (int number: numbers) {
     out.println(number);
}
```

Выводы: в ходе выполнения лабораторной работы была освоена работа с отладчиком IntelliJ.

Контрольные вопросы

1. Зачем нужна отладка?
2. Какие функции предоставляет отладчик IntelliJ?
3. Что такое точка остановки?
4. Как можно взаимодействовать с переменными в окне отладки?

Примеры конспектов уроков

Урок № 1. Знакомство со средой IntelliJ

Тип урока: комбинированный.
Ключевые слова: JDK, Java SE, IntelliJ IDEA.
Цель урока:

- ознакомление с программной платформой Java SE;
- ознакомление с комплектом разработчика JDK;
- ознакомление со средой IntelliJ IDEA и изучение основных инструментов среды.

Планируемые результаты:

предметные: получение навыков настройки и работы в среде IntelliJ, освоение основных инструментов среды.

метапредметные: способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Время реализации: 1 академический час.
Оборудование и материалы: компьютер, презентационное оборудование.

Ход урока

I. *Этап постановки цели и задач урока, мотивации к учебной деятельности* (*10 мин*)

Учитель. Предлагает ученикам начать знакомство с языком Java. Рассказывает о программировании в целом:

- в XXI веке информационные технологии, компьютеры, гаджеты, Интернет — неотъемлемая часть нашей жизни;
- один из путей развития информационных технологий — программирование;
- язык Java является одним из популярнейших языков с широчайшей областью применения;
- Java — типизированный объектно-ориентированный язык программирования;
- Java был разработан компанией Sun Microsystems, впоследствии приобретён компанией Oracle.

Отмечает, что язык Java при этом содержит в себе множество подходов и парадигм, не только объектно-ориентированный, но и процедурный подход. В этом плане самый базовый синтаксис языка, например, при реализации простых алгоритмов принципиально не отличается от синтаксиса, скажем, языка Pascal. Но чем дальше и глубже происходит изучение языка Java, тем больше выявляются не только ОО-особенности, но и сам стиль языка Java. Как и у любого другого языка, есть особенности и подходы программирования, присущие именно языку Java, и они могут быть довольно значительными.

Ученики. Рассказывают, как они себе представляют программирование, какие языки знают, чему хотят научиться, какие у них цели, почему они выбрали именно этот курс.

Учитель. Кратко описывает содержание курса. Рассказывает о знаниях, умениях и навыках, которые должны приобрести учащиеся по окончании курса; как данный курс может помочь ученикам в достижении своих целей.

II. *Этап актуализации знаний и пробного учебного действия* (*5 мин*)

Учитель. Кратко рассказывает об особенностях языка Java и о платформе JDK.

Особенностью языка Java является кроссплатформенность, т. е. возможность выполнения кода в разных средах и системах. Достигается кроссплатформенность за счёт трансляции Java-кода в специальный байт-код и обеспечения выполнения байт-кода для различных архитектур посредством JVM — Java Virtual Machine (виртуальная машина Java).

Рассказывает про различные компьютерные архитектуры, операционные системы, устройства. Отмечает, что Java-код может исполняться в различных операционных системах (Windows 10, Ubuntu Linux, Mac OS и пр.) за счёт исполнения байт-кода в JVM в отличие от языков на базе платформы Microsoft .Net, например C#, VB.Net и пр., для которых перед исполнением программ в различных операционных системах приходится заново компилировать программный код. Подчёркивает, что тем самым единожды скомпилированный Java-код работает везде, где установлена JVM, и поэтому программы на Java применяются в столь многих системах и устройствах.

Отмечает, что следует различать язык Java и платформу (среду) Java. Язык Java пригоден для чтения человеком, а код, поступающий в JVM в виде .class файлов, не предназначен для чтения разработчиком.

Рассказывает о конкурирующих языковых средствах от Microsoft, даёт сопоставление, сравнительный анализ.

Рассказывает подробнее про JDK, JVM и JRE (см. справочные материалы). Демонстрирует на интерактивной доске установку JDK на компьютер.

Ученики. Повторяют процесс скачивания и установки среды на своих компьютерах.

III. *Этап изучения нового материала* (*10 мин*)

Учитель. Кратко рассказывает про среду IntelliJ IDEA.

В качестве среды разработки будет использоваться IntelliJ IDE Community Edition, действующая на основе лицензии Apache 2.0, т. е. лицензии на свободное использование ПО. Первая версия среды появилась в 2001 году и с тех пор среда непрерывно обновляется и эволюционирует. IDE предоставляет массу возможностей и удобных инструментов для быстрого и правильного программирования (подсветка синтаксиса, автозамена, автопредложение кода, организация проектов и прочие возможности редактора кода) и также берёт на себя контроль операций, связанных с созданием, проверкой и компиляцией кода. IntelliJ имеет множество разработанных надстроек и систему их поддержки.

Акцентирует внимание, что современные среды разработки позволяют значительно ускорить и упростить процесс программирования для разработчика.

Демонстрирует процесс установки и настройки среды на презентационном оборудовании.

Ученики. Повторяют процесс установки платформы IntelliJ на своих компьютерах.

Учитель. Рассказывает про типовой проект Java и его структуру в среде IntelliJ. Демонстрирует на интерактивной доске процесс создания консольного Java-проекта. Реализует типовой пример проекта «Hello, World!».

Отмечает, что на данном этапе обучения нужно принять необходимость создания класса Java и метода `main()` как данность, а подробно эта информация будет разобрана на следующих занятиях.

Ученики. Повторяют за учителем и создают проект «Hello, World!» на своих компьютерах. Запускают код.

Учитель. Даёт ученикам задание:

- поменять строку вывода — вместо «Hello, world!» в консоль должно выводиться приветствие ученику вида «Hello, Миша», где Миша — имя ученика, запустить программу;
- добавить ещё одну команду `System.out.println();` и вывести с помощью неё в консоль любой текст.

Ученики. Выполняют задание. Обсуждают результаты.

IV. *Этап проверки понимания и первичного закрепления* (*5 мин*)

Учитель. Объясняет, что среда IntelliJ очень обширна и полностью её функционал можно сравнить с функционалом кабины пилота пассажирского самолёта, поэтому изучение различных аспектов IntelliJ будет проходить сквозным путём через все уроки курса.

Рассказывает, что для работы с Java существуют и другие среды: Eclipse, Netbeans и т. д. Кратко излагает их историю, плюсы и минусы.

Объясняет содержимое лабораторной работы «Знакомство со средой IntelliJ».

Ученики. Выполняют лабораторную работу под контролем учителя. Обсуждают ход работы и результаты.

V. Этап контроля усвоения и коррекции ошибок (5 мин)

Учитель. Задаёт ученикам контрольные вопросы и участвует в их обсуждении.

1. В чём отличие JRE от JDK?
2. Для чего нужна JVM?
3. Зачем нужен байт-код?
4. Опишите процесс создания проекта в IntelliJ.
5. Какие другие среды, кроме IntelliJ, вы знаете?
6. Какой шаблон по умолчанию существует в IntelliJ?
7. В какой папке хранятся исходные коды проекта?
8. Класс с каким названием создаётся по умолчанию в IntelliJ?
9. Какая комбинация клавиш запускает проект?

Ученики. Отвечают на контрольные вопросы и обсуждают их.

VI. *Информация о домашнем задании, инструктаж по его выполнению* (*5 мин*)

Учитель. Дополнительно объясняет структуру проектов IntelliJ, что физически это папки, находящиеся по определённому адресу. Демонстрирует на интерактивной доске соответствующие папки в проводнике. Объясняет, что их можно копировать на внешние носители, при необходимости архивировать и отправлять по почте. Демонстрирует процесс копирования. Импортирует с флешки проект в IntelliJ.

Даёт домашнее задание.

1. Установить JDK.
2. Установить IntelliJ Community Edition.

3. Скопировать созданный в классе ранее проект Java и импортировать его в установленную среду дома.
4. Выполнить задания лабораторной работы «Знакомство со средой IntelliJ».
5. Изучить информацию о компиляторе javac. Понять отличие между использованием javac и IntelliJ при работе с кодом и его выполнением.

Ученики. Записывают задание. Уточняют его содержание.

Учитель. Проверяет, насколько корректно ученики поняли суть домашнего задания.

VII. *Этап рефлексии деятельности на уроке* (*5 мин*)

Учитель. Предлагает учащимся оценить свою деятельность на уроке. Для этого можно воспользоваться различными интерактивными ресурсами, например https://www.mentimeter.com/app. Учащимся предлагается оценить свою деятельность на уроке, выбрав в приложении один из вариантов ответов.

Рис. 82. Интерактивный ресурс для рефлексии деятельности учащихся на занятии

Используемые материалы: лабораторная работа № 1 «Знакомство со средой IntelliJ»; справочные материалы.

Урок № 2. Первые программы на языке Java. Основные операторы

Тип урока: комбинированный.

Ключевые слова: переменная, оператор.

Цель урока: ознакомление с типовыми возможностями языка Java в области работы с переменными и ввода/вывода данных.

Планируемые результаты:

предметные: получение базовых навыков работы с переменными и операторами, вводом/выводом данных.

метапредметные: способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Время реализации: 1 академический час.

Оборудование и материалы: компьютер, презентационное оборудование.

I. *Этап постановки цели и задач урока, мотивации к учебной деятельности* (*5 мин*)

Учитель. Кратко напоминает ученикам материал первого урока и способ вывода данных в консоль. Рассказывает о способах ввода данных. Объясняет, что данные в Java, как и в других языках программирования, хранятся в переменных. На данном уроке рассматриваются переменные примитивных типов. Использует примеры из лабораторных работ № 2 и 3.

Ученики. Обсуждают, что такое переменные и для чего они необходимы.

II. *Этап актуализации знаний и пробного учебного действия* (*10 мин*)

Учитель. Знакомит учеников со способами объявления переменных и вывода их в консоль. Рассказывает об основных типах операторов. Уточняет, что освоение работы с переменными и операторами будет проходить в рамках всего курса. Использует примеры работы с объявлением переменных и присваиванием им значений, примеры работы с операторами из теоретической части лабораторной работы № 2.

Ученики. Создают новый проект IntelliJ. Создают новый класс и в методе `main()` повторяют примеры учителя.

Учитель. Рассказывает об основных операторах и ключевых словах языка Java. Демонстрирует на выбор примеры из лабораторных работ № 2 и 3.

Ученики. Реализуют в методе main() какой-либо из примеров и запускают код.

III. *Этап изучения нового материала* (*15 мин*)

Учитель. Знакомит учеников с классом `Scanner` и особенностями ввода данных через консоль. Рассматривает возможность ввода различных типов данных с помощью ме-

тодов `next()`, `nextInt()`, `nextDouble()` и т. д. Акцентирует внимание на прекращении ожидания ввода пользователем данных после окончания вызова любого из подобных методов. Рассматривает метод `hasNext()` как средство организации бесконечного ожидания ввода пользователем данных. Объясняет про использование метода `break` для досрочного завершения ожидания ввода данных. Рассматривает пример из теоретической части лабораторной работы № 3.

Ученики. Выполняют в методе `main()` пример и запускают код. Обсуждают результаты.

IV. *Этап проверки понимания и первичного закрепления* (*5 мин*)

Учитель. Даёт ученикам задание написать программу, считывающую имя пользователя из консоли и выводящую ответное приветствие, обращаясь к пользователю по имени. Дополнительно обсуждает варианты использования метода `hasNext()` класса `Scanner`.

Ученики. Выполняют задание. Обсуждают результаты.

V. *Этап контроля усвоения и коррекции ошибок* (*5 мин*)

Учитель. Задаёт ученикам вопросы.

1. Какие типы данных можно считывать из консоли?
2. Можно ли использовать класс `Scanner` для создания консольных игр?
3. Какие основные типы переменных представлены в языке Java?
4. Чем операторы сравнения отличаются от операторов присваивания?
5. Что такое ключевое слово в Java?

Ученики. Отвечают на вопросы и обсуждают их.

VI. *Информация о домашнем задании, инструктаж по его выполнению* (*2 мин*)

Учитель. Даёт задание реализовать программу сложения двух чисел со следующим принципом работы: пользователь вводит в консоль друг за другом два числа; программа считывает оба числа и после ввода последнего выводит в консоль результат сложения двух чисел; написать аналогичные программы для вычитания, умножения, деления, нахождения остатка от деления двух чисел.

Ученики. Записывают задание. Уточняют его содержание.

Учитель. Проверяет, насколько корректно ученики поняли суть домашнего задания.

VII. *Этап рефлексии деятельности на уроке* (*3 мин*)

Учитель. Предлагает учащимся оценить свою деятельность на уроке. Для этого можно воспользоваться различными интерактивными ресурсами, например https://www.mentimeter.com/app.

Учащимся предлагается оценить свою деятельность на уроке, выбрав в приложении один из вариантов ответа.

Используемые материалы: лабораторная работа № 2 «Переменные. Операторы»; лабораторная работа № 3 «Ввод данных».

Урок № 3. Классы и объекты

Тип урока: комбинированный.

Ключевые слова: классы, объекты, предметная область, декомпозиция.

Цель урока: ознакомление с базовыми понятиями объектно-ориентированного подхода; ознакомление с объектной декомпозицией предметной области; получение навыков создания классов и их полей в среде IntelliJ.

Планируемые результаты:

предметные: получение знаний об основах объектно-ориентированного подхода в языке Java, получение навыков по созданию простейших классов в среде IntelliJ.

метапредметные: способность анализировать предметную область, выявлять взаимодействия между составными частями простых систем; способность производить простейшую декомпозицию предметной области, соотносить данные, полученные аналитическим путём в результате декомпозиции предметной области с соответствующими структурами языка Java; способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: эстетическое отношение к языкам программирования, осознание их выразительных возможностей; запуск процесса трансформации мировоззрения, соответствующего современному уровню развития информационных технологий; способность к самостоятельному поиску информации, в том числе умение пользоваться справками программ и интернет-поиском; готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности; готовность к изменениям мировоззренческой парадигмы.

Время реализации: 1 академический час.

Оборудование и материалы: компьютер, презентационное оборудование.

Ход урока

I. *Этап постановки цели и задач урока, мотивации к учебной деятельности* (*5 мин*)

Учитель. Напоминает ученикам о первом пробном проекте «Hello, world!» из урока № 1. На его примере показывает, что Java является объектно-ориентированным языком, и в нём всё существует в виде объектов и классов — заготовок, по которым объекты создаются. Рассказывает, что с помощью классов удобно представлять окружающий мир — в виде объектов и взаимодействия между ними.

II. *Этап актуализации знаний и пробного учебного действия* (*10 мин*)

Учитель. Повторяет, что Java является ОО-языком, поэтому в основе его изучения обязательно должны быть ОО-концепции и ОО-парадигма. Проводит краткий обзор и сравнение процедурного и ОО-подхода.

Существует много подходов к пониманию ООП, один из самых простых — «всё является объектами и объекты обмениваются информацией». В основе ОО-подхода находятся классы и объекты. Поэтому во всех предыдущих лабораторных работах необходи-

мо было сначала создать класс, затем метод `main()`, и только после этого можно было написать код. Можно подискутировать на тему, что процедурный подход можно рассматривать как передачу данных, ОО — как передачу данных и объектов.

Процедурные языки применяются в рамках школьного курса информатики, особенно при изучении и реализации алгоритмов. Процедурный подход подразумевает последовательное выполнение операторов. При этом возможно объединять операторы в более крупные блоки, подпрограммы и вызывать при необходимости.

Отмечает, что процедурный подход — это один из видов программирования, применяемый в первую очередь для реализации алгоритмов и управления техническими системами.

Процедурный подход необходим, но недостаточен для удобного описания динамических сложных систем, поэтому он дополняется ОО-подходом, но при этом сам является необходимым «кирпичиком» программ. Например, методы классов обычно наполнены процедурным кодом, без которого невозможно реализовывать алгоритмы.

В коде Java не всегда можно чётко отделить алгоритмический код от ОО-кода, они переплетены вместе, представляют собой гибридную структуру. Например, в обычном последовательном алгоритме могут присутствовать классы или объекты.

Java является ОО-языком, но современная архитектура компьютеров построена на базе фон-Неймановской архитектуры, т. е. программа, по сути, есть изменение состояний хранимых переменных в процессе выполнения последовательности операторов. Поэтому «под капотом» как ОО-языков, так и функциональных всё равно лежит процедурный подход. ООП является своего рода надстройкой, которая позволяет человеку оперировать кодом с помощью понятий и конструкций, которые ближе к человеческому сознанию и его возможностям восприятия окружающей действительности.

Эволюция языков программирования движется в сторону приближения к человеческому мышлению, языку, когнитивным структурам и образам человеческого сознания. Переход от процедурной парадигмы к ОО-парадигме является важным шагом в этом направлении, потому что процедурный подход более присущ сознанию условного робота, шаг за шагом выполняющего алгоритм, а ОО-подход ближе к сознанию человека, потому что оперирует абстрактными концепциями. В ОО-подходе появляются слои кода, слои абстракции, появляются границы, которых нет в процедурном подходе. ООП позволяет программировать в понятиях, максимально приближенных к заданной предметной области.

Для продвинутых учеников можно отметить, что появление лямбда-выражений в Java, начиная с JDK 8.0, привносит элементы функционального подхода и тоже в значительной мере способствует дальнейшей эволюции языка в сторону сближения с человеческим языком, когнитивными структурами и механизмами человека.

Ещё раз акцентирует внимание, почему необходимо создать класс, чтобы вывести строку «Hello, world!» в консоль. Что это особенность языка Java, где первичен класс, и чтобы что-то реализовать, сначала нужно реализовать класс — структуру, содержащую всё остальное.

У многих слово «программирование» ассоциируется именно с процедурным программированием, только с алгоритмами и их реализацией. Важно объяснять и акцентировать внимание, что ОО-подход — это переход на другой уровень оперирования предметной областью, моделирования окружающей реальности. Происходит переход на уровень абстрактных понятий, объектов, которыми описывается программируемый объект или исследуемая система. Можно привести пример, что процедурный подход хорошо подходит для программирования станков с ЧПУ, микроконтроллеров, а если необходимо запрограммировать сложную многообразную систему, состоящую из множества взаимодей-

ствующих и динамически меняющихся компонентов, удобнее сделать это на базе ОО-подхода.

Организует дискуссию: понимание различия между ОО и процедурным подходом очень важно для полноценного изучения языка Java. Большинство знакомится с программированием на основе нулей, единиц, битов, байтов, простых алгоритмов, что приводит к прочной ассоциации программирования с пошаговой реализацией алгоритмов, т. е. только с процедурным подходом. В дальнейшем это может привести к затруднениям в понимании, освоении и применении ОО-концепций на практике. Возможно, идеальным вариантом является параллельное изучение обеих концепций.

Ученики. Участвуют в дискуссии. Обсуждают новые термины и понятия.

III. *Этап изучения нового материала (15 мин)*

Учитель. Предлагает ученикам на листе бумаги описать некоторую предметную область: дом, автомобиль, компьютер, магазин, человека, школу — и попробовать провести декомпозицию на составные части. Например: магазин состоит из склада, торгового зала, продукции. Внутри этих частей могут быть другие активные взаимодействующие части, например: продавцы, покупатели, работники склада. Могут происходить процессы между ними (покупка, продажа, доставка). У частей могут быть свойства: у человека — имя, вес, пол, рост, зарплата. Также могут быть действия, которые они совершают, например, продавец продаёт, покупатель покупает, на склад поступает продукция, сотрудник склада размещает товары на складе, упаковка продукции предоставляет информацию о сроке годности, продавцы размещают продукцию на стеллажах в торговом зале и т. д. Подискутировать на тему: что в рамках рассмотренных частей целого и процессов между ними относится к обычным алгоритмам, которые можно реализовать в методе `main()`, а что удобнее представить в виде системы взаимодействующих объектов. Например: алгоритм расчёта суммы чека на кассе по формуле с учётом скидки хорошо реализуется на базе процедурного подхода. А учёт товаров на полках, транзакции, взаимодействие сотрудников, торгового зала, состоящего из множества компонентов (полки, зоны, кассы, инфраструктура), можно хорошо описать на базе ОО-подхода в виде системы объектов, обменивающихся различными сообщениями. Особенно это удобно при необходимости дальнейшей модификации и поддержки кода.

Акцентирует внимание, что хотя в ОО-подходе предметная область представляется в виде взаимодействующих объектов, но функционирование и взаимодействие объектов определяется также при помощи обычных алгоритмов, просто алгоритмы начинают использовать не только примитивы, но и переменные объектного типа.

Объясняет, что подобное разделение предметной области на части можно назвать объектной декомпозицией. Преподносит тему объектной декомпозиции также и с точки зрения границ.

Есть разные варианты проведения границ, они зависят от целей декомпозиции. Исходя из определения ОО-парадигмы, что всё является объектами, одним из важнейших этапов для применения ОО-подхода является определение объектов и их границ. Без определения границ может быть затруднительно корректно идентифицировать тот или иной функционал с верным объектом или процессом.

Отмечает, что не существует однозначного варианта декомпозиции и описания предметной области. Всё зависит от уровней детализации, уровней абстракции, исходных целей декомпозиции и прочих факторов.

Тот же магазин можно рассматривать не физически, а как объект, предназначение которого — осуществлять процессы закупки и продажи продукции для снабжения населе-

ния. Тогда уже появляются более абстрактные сущности, такие, как отделы: закупка, продажи, логистика, бухгалтерия. Отделы связаны с помещениями: склад, торговый зал, продукция, офисы. С отделами и помещениями связаны сотрудники: складские работники, менеджеры, директора, бухгалтера, продавцы. И все части взаимодействуют друг с другом, с покупателями, поставщиками продукции. Покупатели покупают товары, оставляют деньги, проходят различные операции и пр.

Например, если рассматривается зоопарк, то от того, моделируется ли финансово-отчётная деятельность, либо развлекательная деятельность зоопарка, либо иерархия животных, и будет зависеть итоговая декомпозиция предметной области «Зоопарк». Или в системе «Школа», если рассматривать её с точки зрения учебного процесса, объектами могут быть: ученик, учитель, урок, задания; доска, парта, книга как объекты, обеспечивающие образовательный процесс. Если же школу рассматривать с точки зрения архитектурно-строительных параметров, то объектами уже могут выступать: комната, лестница, фундамент, пролёт, вход, отопление, проводка и т. д.

Организует дискуссию на тему вариантности объектной декомпозиции.

Ученики. Обсуждают объектную декомпозицию. Для предметной области «Зоопарк» приводят примеры вариантности декомпозиции.

Учитель. Даёт задание на тему объектной декомпозиции, в котором требуется уточнить проведённую ранее на бумаге декомпозицию предметной области (например, магазина) и, где возможно, добавить выражения: является (для оценки принадлежности, например: склад является помещением); содержит, включает (для оценки отношений между частями, например: торговый зал включает в себя стеллажи, товары); использует (чтобы выявить процессы между сущностями, например: продавец использует кассу). Далее выявить действия, совершаемые объектами (методы). Выявить свойства частей (размер, цвет и т. д.) и отобразить всё на бумаге в виде усовершенствованной условной схемы. Коллективно обсудить получившиеся сущности (составные части) и связи между ними.

Продвинутым ученикам можно предложить обобщить сущности с учётом иерархии, например, офис и склад являются подклассом помещений и т. д.

Объясняет, что сущности и связи между ними, выявленные в процессе декомпозиции, имеют реализацию в ООП. Сущности становятся классами, свойства сущностей становятся полями классов, действия — методами. Взаимоотношения с другими сущностями отражаются в полях классов, методах, параметрах методов и т. д. Отношение «содержит, включает» может означать наличие одной сущности как поля класса у другой, отношение «использует» может отражаться в параметрах методов.

Отношение «является» означает наследование типа или интерфейса. Наследование и полиморфизм не рассматриваются в рамках данного курса.

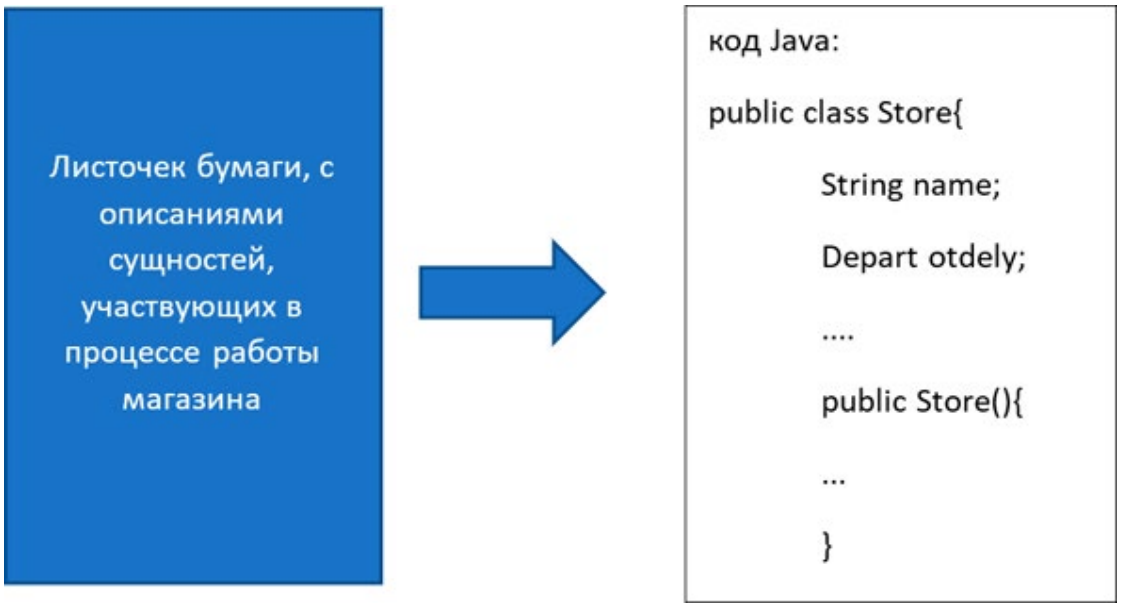

Рис. 83. Описание предметной области может быть переведено в программный код

Учитель. Предлагает обсудить отличие объектной декомпозиции от алгоритмической декомпозиции, в которой требуется уточнение функционала системы и разбиение его на части, при этом неважно, какими объектами он выполняется, какие взаимодействия между объектами при этом происходят. Обсудить соответствие результатов проделанной декомпозиции свойству ООП — «Всё является объектами, и объекты обмениваются сообщениями».

Ученики. Участвуют в дискуссии. Задают вопросы.

IV. *Этап проверки понимания и первичного закрепления* (*5 мин*)

Учитель. Предлагает создать новый проект Java в IntelliJ, далее создать новый тестовый класс `Store`. Предлагает каждому перенести свою схему, составленную на бумаге, в Java-классы проекта IntelliJ и описать на данном этапе только свойства (поля класса). При необходимости даёт краткую информацию по созданию полей класса. Уточняет, что поля класса могут быть как примитивами, так и переменными объектного типа, например, в случае наличия отношения «использует».

Ученики. Создают новые классы для каждой сущности и прописывают поля класса как свойства сущностей.

Учитель. Уточняет, что более детальную информацию для реализации классов они получат после освоения материала лабораторной работы № 5.1 «Классы».

V. Этап контроля усвоения и коррекции ошибок (5 мин)

Учитель. Предлагает учащимся ответить на следующие вопросы.

1. Что такое декомпозиция предметной области?
2. Сколько вариантов декомпозиции одной предметной области может существовать?
3. Что такое класс?
4. Что такое свойства и действия сущности?

Ученики. Отвечают на вопросы, обсуждают их.

VI. *Информация о домашнем задании, инструктаж по его выполнению* (*2 мин*)

Учитель. Даёт ученикам домашнее задание: описать на выбор ещё две любые предметные области. Выявить сущности и их свойства. Создать проект IntelliJ. Согласно выявленным сущностям создать новые классы и их свойства (поля класса).

В дальнейшем после освоения лабораторной работы № 5.1 необходимо будет дополнить классы всех рассмотренных предметных областей методами, учитывая также взаимодействия между сущностями, т. е. отразить отношения между объектами в параметрах методов, в динамических списках, возвращаемых типах методов, полях класса и т. д.

Ученики. Записывают задание. Уточняют его содержание.

Учитель. Проверяет, насколько корректно ученики поняли суть домашнего задания.

VII. *Этап рефлексии деятельности на уроке* (*3 мин*)

Учитель. Предлагает учащимся оценить свою деятельность на уроке. Для проведения данного этапа можно создать интерактивный ресурс, например, в сервисе https://learningapps.org/.

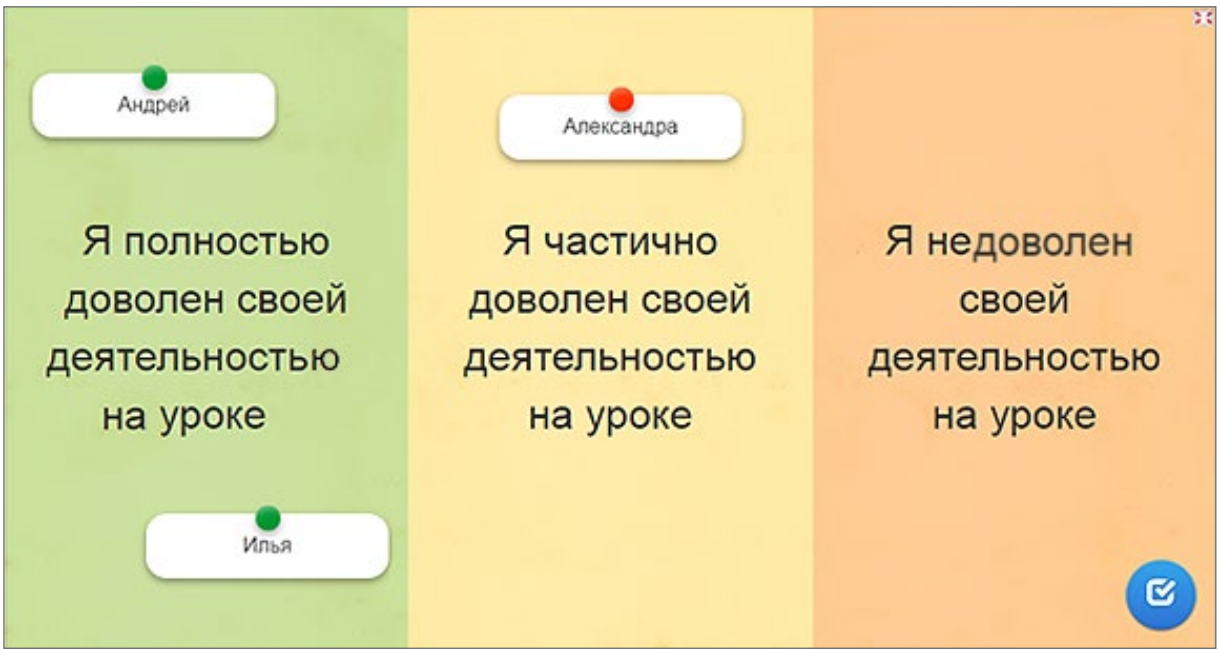

Рис. 84. Ресурс для проведения этапа рефлексии

Используемые материалы: лабораторная работы № 5.1 «Классы».

Урок № 4. Управляющие структуры: ветвление

Тип урока: комбинированный.
Ключевые слова: последовательность, ветвление.
Цель урока: ознакомление с управляющей структурой «ветвление» в языке Java.
Планируемые результаты:
предметные: получение навыков создания последовательных инструкций, применения условных операторов, создания логических выражений.
метапредметные: способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрица-

тельные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Время реализации: 1 академический час.

Оборудование и материалы: компьютер, презентационное оборудование.

Ход урока

I. *Этап постановки цели и задач урока, мотивации к учебной деятельности* (*5 мин*)

Учитель. Обсуждает с учениками определение алгоритма как совокупности последовательных шагов, приводящих к нужному результату. Рассматривает примеры простых алгоритмов, например сложения двух чисел. В этом случае достаточно нескольких инструкций для решения задачи: объявление переменных, их сложение, вывод результата. Задаёт следующие вопросы.

1. Что делать, если появляются дополнительные условия?
2. Всегда ли достаточно последовательного выполнения нескольких инструкций (команд), как в случае сложения двух чисел?

Ученики. Отвечают на вопросы и обсуждают вопросы.

II. *Этап актуализации знаний и пробного учебного действия* (*10 мин*)

Учитель. Рассказывает ученикам про последовательный код и управляющую структуру «ветвление». Приводит простейшие примеры применения и работы с условными операторами из лабораторной работы. Рассматривает простые условия с операторами сравнения. Рассматривает разные варианты условного оператора `if()`. При необходимости рассматривает тернарный оператор (оператор Элвиса).

Ученики. Создают новый проект IntelliJ. Создают новый класс и в методе `main()` повторяют примеры учителя.

Учитель. Объясняет ученикам, что поток управления программы, проходя через условные операторы, тоже выполняет некоторую последовательность шагов. Однако отличие от простой последовательности инструкций в том, что в данном случае возможны различные варианты последовательностей в зависимости от исходных условий (например, значений переменных, используемых в условии).

III. *Этап изучения нового материала* (*15 мин*)

Учитель. Рассказывает ученикам про логические выражения и демонстрирует таблицу логических операций. Обсуждает с учениками, в каких ситуациях применяется логическое И, логическое ИЛИ и логическое НЕ. В данном уроке исключающее ИЛИ рассматривать необязательно. Приводит простые примеры из лабораторной работы. Объясняет, что с помощью логических выражений можно формировать сложные составные условия.

Ученики. Обсуждают логические операторы и варианты их применения. Реализуют в созданном проекте примеры и запускают код.

IV. *Этап проверки понимания и первичного закрепления* (*5 мин*)

Учитель. Объясняет ученикам, что для хорошего усвоения данной темы необходимо её закрепление на практике, и описывает содержание лабораторной работы № 4. Задаёт ученикам следующие вопросы.

1. Какие существуют варианты условного оператора `if()`?
2. Для чего нужны условия?
3. Чем логическое И отличается от логического ИЛИ?

Ученики. Обсуждают пройденный материал. Отвечают на вопросы учителя.

V. *Этап контроля усвоения и коррекции ошибок* (*5 мин*)

Учитель. Даёт ученикам задание реализовать одну из следующих программ, используя синтаксис языка Java, но на бумаге, без привлечения среды IntelliJ.

1. Найти наибольшее из двух чисел *a* и *b* и вывести его в консоль.
2. Найти наименьшее из двух чисел *a* и *b* и вывести его в консоль.
3. Из трёх чисел *a*, *b*, *c* найти наименьшее и вывести его в консоль.

Ученики. Реализуют алгоритм. Обсуждают результаты. Переводят код в среду IntelliJ и разбирают синтаксические ошибки.

VI. *Информация о домашнем задании, инструктаж по его выполнению* (*2 мин*)

Учитель. Даёт домашнее задание.

1. Выполнить пункты 5 и 6 из лабораторной работы № 4.
2. Ответить на контрольные вопросы из лабораторной работы № 4.

Ученики. Записывают задание. Уточняют его содержание.

Учитель. Проверяет, насколько корректно ученики поняли суть домашнего задания.

VII. *Этап рефлексии деятельности на уроке* (*3 мин*)

Учитель. Предлагает учащимся оценить свою деятельность на уроке. Для этого можно воспользоваться различными интерактивными ресурсами, например https://www.mentimeter.com/app.

Учащимся предлагается оценить свою деятельность на уроке, выбрав в приложении один из вариантов ответов.

Используемые материалы: лабораторная работа № 6 «Управляющие структуры».

Урок № 5. Управляющие структуры: циклы

Тип урока: комбинированный.
Ключевые слова: цикл.
Цель урока: ознакомление с управляющей структурой «цикл» в языке Java.
Планируемые результаты:
предметные: получение навыков создания и применения циклов в языке Java.
метапредметные: способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в

рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Время реализации: 1 академический час.

Оборудование и материалы: компьютер, презентационное оборудование.

I. *Этап постановки цели и задач урока, мотивации к учебной деятельности* (*5 мин*)

Учитель. Напоминает ученикам материал прошлого урока, связанный с ветвлениями. Ставит вопрос: что делать, если необходимо многократное повторение каких-либо операций или необходимо одинаковым образом обработать тысячи различных чисел? Всегда ли достаточно последовательного выполнения нескольких инструкций (команд), как в случае сложения двух чисел? Или условий, как в случае сравнения двух чисел? Приводит пример с игровым полем из лабораторной работы № 6.

Ученики. Обсуждают вопросы.

II. *Этап актуализации знаний и пробного учебного действия* (*10 мин*)

Учитель. Разбирает цикл `for()` и его структуру. Приводит простейшие примеры применения и работы с циклом `for()` из лабораторной работы. Рассматривает разные варианты инициализации начального значения, условия и шага в цикле `for()`.

Ученики. Создают новый проект IntelliJ. Создают новый класс и в методе `main()` повторяют примеры учителя.

Учитель. Объясняет ученикам, что поток управления программы, проходя через цикл, выполняет некоторую последовательность шагов. Однако отличие от простой последовательности инструкций в том, что в данном случае возможны различные варианты последовательностей в зависимости от исходных условий (например, может меняться количество циклов или переменная условия, или шаг цикла `for()`).

III. *Этап изучения нового материала* (*15 мин*)

Учитель. Рассматривает реализацию цикла `while()`. Приводит примеры из лабораторной работы. Рассматривает общие свойства и различия циклов `for()` и `while()`. Объясняет, когда удобнее применять тот или иной цикл. Обсуждает с учениками применение бесконечных циклов и варианты их реализации посредством `for()` и `while()`.

Ученики. Обсуждают циклы `for()` и `while()`. Реализуют в методе `main()` примеры и запускают код.

IV. *Этап проверки понимания и первичного закрепления* (*5 мин*)

Учитель. Рассказывает о возможности вложения циклов друг в друга. Приводит пример двойного цикла `for()` из лабораторной работы.

Ученики. Обсуждают материал. Реализуют в методе `main()` пример и запускают код.

V. *Этап контроля усвоения и коррекции ошибок* (*5 мин*)

Учитель. Задаёт ученикам следующие вопросы.

1. В каких случаях могут понадобиться циклы?

2. Для чего могут понадобиться двойные циклы?
3. Можно ли в цикл `for()` вложить цикл `while()`?
4. Для чего нужен бесконечный цикл?
5. В чём различия циклов `for()` и `while()`?

Ученики. Обсуждают пройденный материал. Отвечают на вопросы.

Учитель. Рассказывает про возможность выявления ошибок в коде с помощью отладки. Рассказывает про средства отладки IntelliJ. Показывает меню отладчика.

VI. *Информация о домашнем задании, инструктаж по его выполнению* (*2 мин*)

Учитель. Рассказывает об операторах break и continue и даёт следующее задание.

1. Проработать примеры с использованием данных операторов из лабораторной работы.
2. Ответить на контрольные вопросы из лабораторной работы № 6.

Ученики. Записывают задание. Уточняют его содержание.

Учитель. Проверяет, насколько корректно ученики поняли суть домашнего задания.

VII. *Этап рефлексии деятельности на уроке* (*3 мин*)

Учитель. Предлагает учащимся оценить свою деятельность на уроке. Для этого можно воспользоваться различными интерактивными ресурсами, например https://www.mentimeter.com/app.

Учащимся предлагается оценить свою деятельность на уроке, выбрав в приложении один из вариантов ответов.

Используемые материалы: лабораторная работа № 6 «Управляющие структуры»; лабораторная работа № 10 «Отладка кода».

Урок № 6. Структуры данных

Тип урока: комбинированный.
Ключевые слова: массив, список.
Цель урока: ознакомление с массивами и списками в языке Java.
Планируемые результаты:
предметные: получение навыков создания и применения массивов и динамических списков в языке Java.

метапредметные: способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.

личностные: готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.

Время реализации: 1 академический час.
Оборудование и материалы: компьютер, презентационное оборудование.

I. *Этап постановки цели и задач урока, мотивации к учебной деятельности* (*5 мин*)

Учитель. Напоминает ученикам материал урока, связанного с циклами. Приводит пример: необходимо рассмотреть 100 разных переменных и провести над ними какие-то одинаковые операции: например, вывести их в консоль. Разве не удобно было бы использовать цикл? Но как его применить к разным переменным? Как можно рассмотреть сотни переменных в рамках одной структуры? Объясняет, что в данном случае можно воспользоваться массивами или списками. Приводит примеры из лабораторной работы.

Ученики. Обсуждают вопросы учителя.

Используемые материалы: лабораторная работа № 7 «Массивы»; лабораторная работа № 9.1 «Списки»; лабораторная работа № 9.2 «Двумерные списки».

II. *Этап актуализации знаний и пробного учебного действия* (*10 мин*)

Учитель. Рассматривает объявление и инициализацию массива, обращение к элементам, заполнение и вывод массива в консоль. Рассматривает объявление и инициализацию списка, вывод элементов списка в консоль. Сравнивает массивы и списки. Приводит примеры из лабораторной работы.

Ученики. Создают новый проект IntelliJ. Создают новый класс и в методе `main()` повторяют примеры учителя.

III. *Этап изучения нового материала* (*15 мин*)

Учитель. Рассматривает реализацию двумерных массивов. Обсуждает с учениками, в каких ситуациях применяются двумерные массивы. При необходимости упоминает об индексации вида $i \cdot m+j$. Рассматривает вывод элементов динамического списка посредством цикла foreach или итератора. Использует примеры из лабораторных работ.

Ученики. Реализуют в методе `main()` примеры учителя и запускают код. Обсуждают результаты.

IV. Этап проверки понимания и первичного закрепления (*5 мин*)

Учитель. Рассказывает про параметризованные типы в списках. Приводит пример из лабораторной работы.

Ученики. Обсуждают материал. Выполняют в методе `main()` пример и запускают код.

V. *Этап контроля усвоения и коррекции ошибок* (*5 мин*)

Учитель. Задаёт ученикам следующие вопросы.

1. В чем различия массивов и списков?
2. Для чего могут понадобиться двумерные массивы?
3. Можно ли менять размерность массива?
4. В чём удобство списка?
5. В чём удобство цикла типа foreach?

Ученики. Обсуждают пройденный материал. Отвечают на вопросы.

VI. *Информация о домашнем задании, инструктаж по его выполнению* (*2 мин*)

Учитель. Даёт следующее задание.

1. Проработать все примеры из теоретических частей лабораторных работ № 7, 9.1, 9.2.
2. Ответить на контрольные вопросы из лабораторных работ № 7, 9.1, 9.2.

Ученики. Записывают задание. Уточняют его содержание.

Учитель. Проверяет, насколько корректно ученики поняли суть домашнего задания.

VII. *Этап рефлексии деятельности на уроке* (*3 мин*)

Учитель. Предлагает учащимся оценить свою деятельность на уроке. Для этого можно воспользоваться различными интерактивными ресурсами, например https://www.mentimeter.com/app.

Учащимся предлагается оценить свою деятельность на уроке, выбрав в приложении один из вариантов ответа.

Используемые материалы: лабораторная работа № 7 «Массивы»; лабораторная работа № 9.1 «Списки»; лабораторная работа № 9.2 «Двумерные списки».

Урок № 7. Отладка кода

Тип урока: комбинированный.
Ключевые слова: отладка, точка остановки, режим отладки.
Цель урока: ознакомление с возможностями отладки кода в среде IntelliJ.
Планируемые результаты:
предметные: получение навыков отладки кода и коррекции ошибок в среде IntelliJ.
метапредметные: способность ставить и формулировать для себя цели действий, прогнозировать результаты, анализировать их (причём как положительные, так и отрицательные), делать выводы в процессе работы и по её окончании, корректировать намеченный план, ставить новые цели; умение соотносить свои действия с планируемыми результатами, осуществлять контроль своей деятельности, определять способы действий в рамках предложенных условий, корректировать свои действия в соответствии с изменяющейся ситуацией; умение оценивать правильность выполнения учебной задачи.
личностные: готовность и способность обучающихся к саморазвитию и личностному самоопределению; сформированность их мотивации к обучению и целенаправленной познавательной деятельности.
Время реализации: 1 академический час.
Оборудование и материалы: компьютер, демонстрационное оборудование.

I. *Этап постановки цели и задач урока, мотивации к учебной деятельности* (*5 мин*)

Учитель. Рассказывает ученикам про отладку кода и про отладчик IntelliJ. Объясняет, что с помощью отладки намного проще отследить ошибки, чем пытаться найти их вручную или только с помощью вывода вспомогательной информации в консоль. Рассказывает об основных функциях отладчиков.

II. *Этап актуализации знаний и пробного учебного действия* (*10 мин*)

Учитель. Знакомит учеников с меню отладчика в среде IntelliJ. Рассказывает о точках остановки, запуске режима отладчика, функциональных клавишах *F7*, *F8*, меню отладчика.

Ученики. Открывают предыдущий проект IntelliJ. Создают точки остановки и проходят любой реализованный ранее алгоритм в режиме отладчика.

Учитель. Объясняет ученикам, что с помощью отладчика можно проследить за потоком управления программы. Разбирает удобство пошагового отслеживания переменных в режиме отладчика.

III. *Этап изучения нового материала* (*15 мин*)

Учитель. Разбирает пример из теоретической части лабораторной работы №10.

Ученики. Выполняют в методе `main()` пример учителя и запускают код. Обсуждают результаты.

IV. *Этап проверки понимания и первичного закрепления* (*5 мин*)

Учитель. Рассказывает про возможность отслеживания значений переменных и массивов наведением курсора мыши. Рассказывает про досрочное завершение отладки. Даёт ученикам следующее задание: с помощью отладчика IntelliJ найти ошибку в следующем коде (программа должна выводить в консоль нечётные числа среди чисел от 1 до 10).

```
public static void main(String[] args) {
     int number=10;
     for(int i=0; i<=number; i+=2){
          if(i%2==1) {
               System.out.println("Нечётное число:"+i);
               i++;
          }
     }
}
```

Ученики. Обсуждают пройденный материал. Выполняют задание.

V. *Этап контроля усвоения и коррекции ошибок* (*5 мин*)

Учитель. Задаёт ученикам следующие вопросы.

Для чего нужна досрочная остановка отладки?

В чём различие режимов прохода по строкам в случае захода внутрь методов и без захода?

В чём удобство использования отладчика?

Ученики. Обсуждают пройденный материал. Отвечают на вопросы.

VI. *Информация о домашнем задании, инструктаж по его выполнению* (*2 мин*)

Учитель. Даёт следующее задание.

Проработать с помощью отладчика ошибки в программах из предыдущих уроков и лабораторных работ.

Ответить на контрольные вопросы из лабораторной работы №10.

Ученики. Записывают задание. Уточняют его содержание.

Учитель. Проверяет, насколько корректно ученики поняли суть домашнего задания.

VII. ***Этап рефлексии деятельности на уроке*** **(*3 мин*)**

Учитель. Предлагает учащимся оценить свою деятельность на уроке. Для этого можно воспользоваться различными интерактивными ресурсами, например https://www.mentimeter.com/app.

Учащимся предлагается оценить свою деятельность на уроке, выбрав в приложении один из вариантов ответа.

Используемые материалы: лабораторная работа № 10 «Отладка кода».

Форма аттестации, примеры контрольно-оценочных материалов

Во время проведения курса предполагаются текущий, промежуточный и итоговый контроль.

Текущий контроль осуществляется регулярно во время проведения каждого лабораторного занятия. Он заключается в ответах учащихся на контрольные вопросы, демонстрации учащимися разработанного кода в IntelliJ, фронтальных опросов учителем.

Также в тематическом планировании предполагается две промежуточные контрольные работы.

Контрольная работа для проверки полученных навыков по темам «Управляющие структуры»

1. Найти расстояние между двумя точками, заданными с помощью координат на плоскости.
2. Вычислить значение функции $y(x) = x^2 - 7x + 8$ для заданного с клавиатуры значения аргумента x.
3. Определить количество положительных чисел среди трёх введённых в консоль.
4. Реализовать метод, параметром которого является целое число — количество лет, а возвращаемым значением — число месяцев, содержащихся в этом количестве лет.
5. Дан двумерный массив `int[][] a = new int[6][6]`.

Реализовать алгоритм, заполняющий его следующим образом.

```
1 1 1 0 0 0
1 1 1 0 0 0
1 1 1 0 0 0
1 1 1 0 0 0
1 1 1 0 0 0
1 1 1 0 0 0
```

Контрольная работа для проверки полученных навыков по теме «Списки», «Классы»

1. Создать класс `Tank` с нужными полями (обязательны поля `String name` — название танка и `int power` — мощность орудия) и методами (как минимум нужны методы `fire()` — стрелять, `move()` — двигаться, `stop()` — остановиться). В каждом методе также реализовать вывод нужной информации в консоль через

`System.out.println()` (у метода `fire()` выводимая информация будет зависеть от переменной `power`). В методе `main()` проверить создание классов и работу методов. Можно добавить управление танком через консоль: считывать вводимые значения и давать танку команды, например: `fire()`, `move()`, `stop()`, `getTechnicalInfo()`.

2. Создать список `ArrayList<Tank> tanks`. Заполнить список 100 танками с помощью цикла `for()`. Перебрать все танки через цикл типа foreach. При переборе вызывать у каждого танка метод `fire()`.
3. Даны два списка `ArrayList<String>`. Создать третий список размерности наибольшего списка, где каждый элемент равен конкатенации соответствующих значений первых двух.
4. Сгенерировать два динамических списка типа `ArrayList`, заполненные случайными целыми числами (использовать класс `Random`). Создать третий массив, включая в него только те элементы, которые встречаются в исходных массивах только один раз.
5. Реализовать запись значений, введённых пользователем в консоль, в список.

Критерии оценки

Можно предложить следующие критерии оценивания контрольных заданий.

Набранный балл	Оценка	Критерий
5	Высокий уровень	Программа написана правильно и корректно, получен верный результат выполнения. Приведено полное обоснование выбора алгоритма для реализации программы. Получены верные ответы на дополнительные вопросы преподавателя
4	Средний уровень	Программа написана правильно и корректно, получен верный результат выполнения. Приведено полное обоснование выбора алгоритма для реализации программы. Не получены или получены неполные ответы на дополнительные вопросы преподавателя
3	Низкий уровень	В программном коде содержатся ошибки или программа выводит неверный результат. Не приведено или приведено неполное обоснование выбора алгоритма. Не получены или получены неполные ответы на дополнительные вопросы преподавателя

Итоговый балл можно рассчитывать как среднее арифметическое баллов за каждое задание контрольной работы.

Материалы для организации и проведения учебно-исследовательской и проектной деятельности школьников

Проекты по программированию представляют собой проекты, результатами которых является программа для решения той или иной задачи. Особенностью является то, что одна и та же задача в зависимости от уровня проработки может быть решена как начинающим, так и опытным программистом.

При выполнении проекта по программированию учащиеся имеют следующие возможности: получить умения самостоятельно формулировать цели и задачи проекта; планировать свою деятельность; повысить уровень программирования на языке Java; получить умение представления результатов своей деятельности.

Проект может разрабатываться индивидуально или группой учащихся. Если задача достаточно сложная, то проект может быть разбит на подзадачи, подпроекты. Каждую подзадачу будут выполнять различные группы участников проекта. Например, одна группа будет заниматься разработкой алгоритма, другая группа — непосредственно написанием и отладкой кода, третья — подготовкой к презентации проекта.

Этапы работы над проектом по программированию могут совпадать с этапами разработки программ, представленными на следующем рисунке.

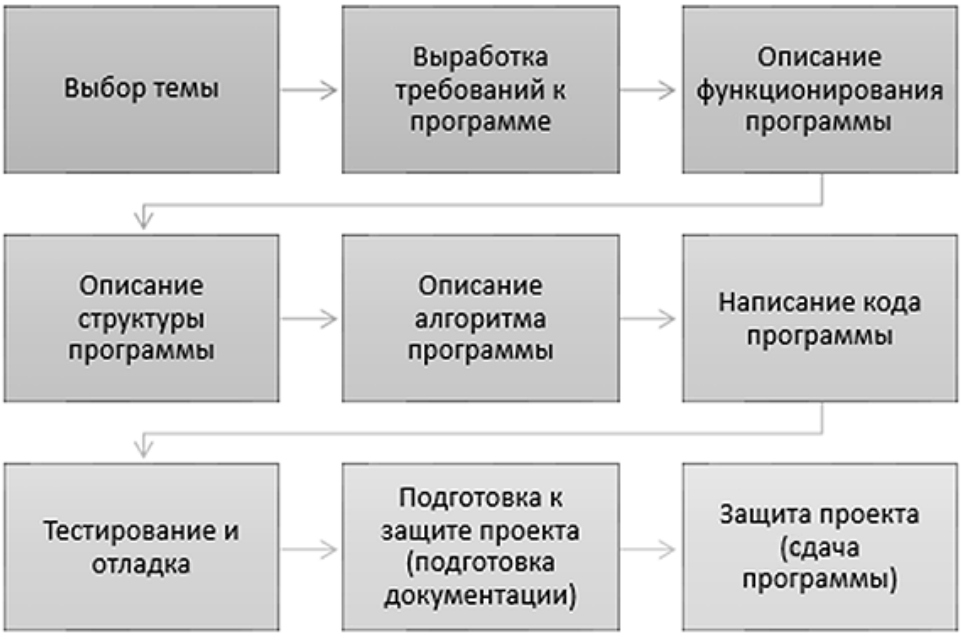

Рис. 85. Этапы работы над проектом

Структура проекта по программированию на языке Java

В помощь участникам проекта можно предложить заполнить следующий учётный лист.

Тема проекта:
Творческое название (при наличии):
Основополагающий вопрос:
Авторы:
1.

2.
3.
...
Предметная область:
Краткая аннотация:
Этапы выполнения проекта:

При подготовке к защите проекта учащимся необходимо подготовить презентацию и доклад, в котором отражаются основные этапы разработки программы, представлен алгоритм решения задачи, листинг программы, основные результаты работы. В помощь учащимся можно предложить заполнить следующий чек-лист.

1. Аннотация.
2. Содержание.
3. Постановка задачи:
   а) возможности использования программы;
   б) описание интерфейса.
4. Формализация алгоритма:
   а) перечень подпрограмм (при наличии);
   б) описание алгоритма (блок-схема или подробное словесное описание алгоритма).
5. Листинг программы.
6. Тестовые примеры:
   а) результаты работы;
   б) скриншоты результатов работы.
7. Описание размещения.
8. Требования к программным и аппаратным средствам.

Для оценивания проекта могут быть разработаны специальные оценочные листы. Ниже представлен пример оценочного листа.

Лист оценивания проекта

Критерий оценивания	1-я группа	2-я группа	...
Актуальность темы			
Соответствие содержания проекта заявленной теме			
Техническая сложность разработанной программы			
Оригинальность алгоритма			
Дизайн интерфейса			
Степень разработанности программы			
Применение программы для решения аналогичных задач			
Итоговое количество баллов			

Ниже приведены возможные темы исследовательских проектов учащихся.

Проект 1. «Угадай число»

1. Написать игру «Угадай число» по следующему техническому заданию.

Создать класс «GuessNumber» и в методе `main()` сгенерировать случайное число *x* от 0 до 99.

Реализовать ввод пользователем вариантов ответа с помощью класса `Scanner`.

В зависимости от разницы *diff* между введённым и загаданным числом выводить в консоль:

- *diff*≥30: «Очень холодно!»
- *diff*≥20: «Холодно!»
- *diff*>10: «Прохладно...»
- *diff*≤10: «Тепло!»
- *diff*≤5: «Горячо!»
- *diff*≤3: «Очень горячо!»

Если пользователь хочет досрочно прекратить игру, то может воспользоваться клавишей *Q*, иначе игра идёт до верного ответа пользователя.

В случае верного ответа в консоль выводится сообщение: «Число угадано! Количество попыток: 30», где 30 — общее число попыток.

2. Пользуясь материалами лабораторной работы № 5.1 «Классы», перенести код из метода `main()` в следующие статические методы.

```
  public static int getRandom(int range){ /*возвращает случайное
число из диапазона range*/ }
  public static String getDistance(int userNumber, int gameNumber)
{ /*возвращает соответствующее текстовое сообщение в зависимости
от разницы между вариантом пользователя и загаданным числом*/ }
  public static void mainGameLogics(String name){ /*реализует
каркас логики игры: ввод имени пользователя, генерацию числа (те-
перь уже через вызов метода getRandom()), ожидание ввода вариан-
тов пользователя, генерацию ответов (теперь через вызов метода
getDistance()), и т.д.*/ }
```

При необходимости создать прочие методы.

Вызвать в методе `main()` класса `GuessNumber` метод `mainGameLogics()` и запустить игру, проверить её работоспособность.

Обсудить, имеет ли смысл сделать все эти методы нестатическими, т. е. методами объекта? Возможно, следует создать конструктор и дополнительные поля класса.

3. Проанализировать блок условий:

- *diff*≥30: «Очень холодно!»
- *diff*≥20: «Холодно!»
- *diff*>10: «Прохладно...»
- *diff*≤10: «Тепло!»
- *diff*≤5: «Горячо!»
- *diff*≤3: «Очень горячо!»

Заметить, что он завязан на конкретные числа — 30, 20 и т. д. В этом случае, если сменить максимальное значение загадываемого числа, то игра будет неполноценной. Подумать, какую формулу можно использовать в блоке условий, чтобы код стал универсальным и не был привязан только к значению 100. Например, если максимальное число 1000, то и числа в условии должны пропорционально увеличиться.

Проект 2. «Морской бой»

Реализовать консольную игру «Морской бой».

Использовать двумерные матрицы вида `int[][]` для хранения данных. Для взаимодействия с системой использовать класс `Scanner` и вывод в консоль.

Проект 3. «Графика в Java»

1. Изучить возможности библиотеки Java.AWT для создания изображений типа BufferedImage и типы графических элементов, реализуемых в графическом контексте `Graphics2D`. Реализовать класс с методами, генерирующими изображения.

Примерный код для генерации изображений может быть следующим.

```
package com.itcube;

import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.geom.Ellipse2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class ProjectImage {
     private static BufferedImage drawGraphics() {
          BufferedImage imgBuff = new BufferedImage(500, 200,
BufferedImage.TYPE_INT_RGB);
          Graphics2D g2 = imgBuff.createGraphics();
     g2.setFont(new Font("Serif", Font.ITALIC, 48));
     g2.setColor(Color.RED);
     g2.drawString("Hello, ITCube!", 100, 100);
     g2.setColor(Color.YELLOW);
     g2.setStroke(new BasicStroke(1.0f,
          BasicStroke.CAP_BUTT,
          BasicStroke.JOIN_MITER,
          10.0f, new float[]{10.0f}, 0.0f));
     g2.drawOval(30,30,50,50);
     g2.setStroke(new BasicStroke());
     g2.setColor(Color.red);
     g2.drawRect(70,30, 30,30);
     g2.setPaint(new GradientPaint(250,100,Color.GREEN,350,
150,Color.YELLOW));
     g2.fill (new Ellipse2D.Double(250, 100, 100, 50));
     g2.dispose(); return imgBuff;
  }
  public static void main(String[] args) {
     BufferedImage image=drawGraphics();
     File MyImage = new File("image.jpeg");
     try {
          MyImage.createNewFile();
          ImageIO.write(image, "jpeg", MyImage);
     } catch (IOException e) {
```

```
            e.printStackTrace();
            }
        }
    }
```

В данном примере метод drawGraphics() создаёт изображение, а в методе main() происходит запись следующего изображения в папку проекта.

Рис. 86. Изображение, генерируемое программой

2. Реализовать настройку изображения через параметры метода, генерирующего изображение `drawGraphics()`. Например, задавать текст, вставляющийся в изображение, параметры фигур, текстовые эффекты и т. д. Подключить интерактивный диалог с пользователем посредством класса `Scanner`.

Источники информации

1. Документация к языку Java: https://docs.oracle.com/javase/tutorial/java/concepts/index.html
2. Стек и куча в Java: https://topjava.ru/blog/stack-and-heap-in-java/
3. Классы в языке Java: https://docs.oracle.com/javase/tutorial/java/concepts/class.html
4. Спецификация к Java SE: https://docs.oracle.com/javase/specs/
5. Спецификация к языку Java: https://docs.oracle.com/javase/specs/jls/se16/html/index.html
6. Java. Новое поколение разработки / Бенджамин Э. — СПб.: Питер, 2014. — 560 с.
7. Различия между Java SE и Java EE: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html

Содержание

Григорьев Сергей Георгиевич
Сабитов Рустэм Адиевич
Смирнова Гульнара Сергеевна
Сабитов Шамиль Рустэмович

Реализация дополнительной общеобразовательной программы
по тематическому направлению «Программирование на языке Java»
с использованием оборудования центра цифрового образования детей «IT-куб»

Методическое пособие

Центр Естественно-научного и математического образования
Руководитель Центра *З. Г. Гапонюк*
Ответственный за выпуск *М. Д. Полежаева*
Редактор *М. Д. Полежаева*
Художественное оформление *Т. В. Глушкова*
Компьютерная вёрстка и техническое редактирование *Н. А. Разворотнева*
Корректор *Н. А. Смирнова*