Программирование
1 семестр

ИТМО

Модули и инструменты разработки

Проблемы организации кода

- Сложно иметь много классов без структуры
- Возможные конфликты имен
- Контроль доступа к классам и элементам классов

Организация кода Java

ИТМО

- До Java 9 - packages
  - Логические группы классов
  - Нет контроля зависимостей
  - public классы видны всем

- Java 9+ - modules
  - Архитектурные группы модулей
  - Явное указание зависимостей
  - Ограничение доступа к классам

Пакеты — Packages

ИТМО

- Пакет — логическая группа классов и интерфейсов
  - Пакет формирует пространство имен
    - Имена классов в пакете должны быть уникальными
    - Имя пакета в приложении должно быть уникальным
  - Пакеты могут быть вложенными — иерархическая структура
    - обратный домен: пакет `ru.itmo.pikt` для домена pikt.itmo.ru
  - Стандартные модификаторы доступа

Пакеты в коде и в файловой системе

- Ключевое слово package в первой строке исходного кода
- Полное имя класса — package.ClassName
  - ru.itmo.pikt.myproject.MyClass
- Пакеты соответствуют каталогам
  - для исходного кода (желательно)
  - для готовых классов (обязательно)

```
package ru.itmo.pikt.myproject;

public class MyClass {  }
```

```
.
└── ru
    └── itmo
        └── pikt
            └── myproject
                └── MyClass.class
```

Импорт классов

ИТМО

- Позволяет использовать короткое имя класса
  - import ru.itmo.pikt.myproject.MyClass;
  - import ru.itmo.pikt.myproject.*;
- Импорт не включает подпакеты
  - ~~import ru.itmo.pikt.*;~~

Статический импорт

ИТМО

- Позволяет импортировать статические элементы класса и обращаться к ним по короткому имени (без имена класса)
  - import static java.lang.Math.PI;
  - import static java.lang.Math.*;

Пакеты в Java API

- Пакеты стандартной библиотеки: java.* и javax.*
  - java.lang — автоматически импортируется
  - java.util — утилиты (Scanner, Date)
  - javax.swing - графическая библиотека Swing
  - и т. д.

Classpath — список путей для поиска

ИТМО

- Classpath — список путей, где лежат классы и ресурсы
  - каталоги
  - JAR-архивы
- Задается
  - переменной окружения CLASSPATH
  - флагом -cp или -classpath при компиляции и запуске

```
javac -cp classes:/usr/java/lib Hello.java

java -cp C:\Programs\jdk25\lib;D:\java\out mypackage.Main
```

JAR-архивы

- JAR — архив с классами и ресурсами
  - формат ZIP
  - способ распространения приложений
  - содержит файл манифеста
    - META-INF/MANIFEST.MF
  - может быть исполняемым
    - Main-Class: ru.itmo.pikt.prog.app.Main
    - java -jar myapp.jar.

JAR-архивы и пакеты

ИТМО

- JAR и пакеты не зависят друг от друга
  - в одном JAR много пакетов
  - в одном JAR один пакет
  - в нескольких JAR один пакет
  - в нескольких JAR много пакетов
- JAR Hell
  - используется класс, **который нашелся первым**

Доступ к классам и элементам классов

ИТМО

- Базовые модификаторы доступа
  - `public` — доступен всем
  - `protected` — доступен в том же пакете + наследникам
  - ("package-private") — доступен только в том же пакете
  - `private` — доступен только внутри класса

Система модулей Java

ИТМО

- JPMS — Java Platform Module System
- Решает проблемы больших приложений:
  - "JAR Hell"
  - контроль доступа к классам
  - управление зависимостями
- Модуль — группа пакетов и данных
  - module-info.java — описание зависимостей и правил доступа

module-info.java

ИТМО

- **module** *module* { ... }
  - **requires** *module*; — зависимость от другого модуля
  - **exports** *package*; — экспортирование пакета другим модулям
    - экспортированные пакеты
      - public классы доступны зависимым модулям (которые requires)
    - не экспортированные пакеты
      - public классы доступны только внутри модуля

Дополнительные директивы

ИТМО

- Кроме основных requires и exports:
  - requires transitive *module* — зависимость от другого модуля не только данного модуля, но и зависимых от него
  - requires static *module* — зависимость только на этапе компиляции
  - exports *package* to *modules* — экспорт только указанным модулям

Доступ к закрытым элементам

ИТМО

- До Java 9 можно было получать доступ к private полям и методам с помощью рефлексии
- Доступ в системе модулей к private полям во время выполнения
  - opens *package* — открывает доступ к пакету всем
  - opens *package* to *modules* — только указанным модулям
  - opens *module* — открывает доступ ко всем пакетам модуля

Пример module-info.java

```
module ru.itmo.pikt.mylib {
  requires java.sql;
  requires transitive com.google.gson;
  requires static com.openai.megapack;

  exports ru.itmo.pikt.myapp.config;
  exports ru.itmo.pikt.myapp.db to ru.itmo.pikt.myapp.util;

  opens ru.itmo.pikt.myapp.config to ru.itmo.pikt.myapp.init;
}
```

class-path и module-path

ИТМО

- class-path
  - Список мест, где искать классы
  - Все классы из каталогов и JAR-архивов принадлежат одному безымянному модулю
- module-path
  - Список мест, где искать модули
  - Все JAR-архивы без module-info становятся автоматическими модулями (1 JAR — 1 модуль)
  - Все JAR-архивы с module-info будут именованными модулями

Типы модулей

ИТМО

- Автоматические модули
  - module-path
  - имя JAR-архива без версии
  - видит все в других модулях
  - раздает все свои пакеты
  - во время миграции
- Безымянный модуль
  - class-path
  - видит все в других модулях
  - не раздает свои пакеты
  - обратная совместимость
  - простые проекты
- Именованные модули
  - module-path
  - полный контроль над доступом и зависимостями
  - новые приложения
  - не видят код в безымянных модулях

Менеджеры зависимостей и средства сборки

ИТМО

- Подключать вручную библиотеки неудобно
- Средства для сборки кода и управления зависимостями
  - Maven : pom.xml
  - Gradle : build.gradle
- Автоматически скачивают библиотеки и их зависимости
- Готовые репозитории (Maven Central)

Java Modules и системы сборки

ИТМО

- Зависимости для системы сборки — внешние
  - Что и какой версии нужно скачать и сделать доступным
  - Система сборки знает, где это найти и куда положить
- Зависимости в module-info — внутренние
  - Что нужно модулю для его работы (без уточнения версий)
  - Система модулей не связана с системой сборки
- Конфигурацию задает разработчик

Зачем нужна документция?

- Код читается чаще, чем пишется
- Документация объясняет — что и зачем?
- Описание контракта API для пользователей классов

Javadoc

- Javadoc — стандартное средство документации Java
  - Документация генерируется автоматически в формате HTML из специальных комментариев
  - Документация хранится вместе с исходниками
  - Поддерживается основными IDE
  - Документируется основные элементы: классы, интерфейсы, методы, конструкторы, поля

Структура Javadoc-комментария

- `/** Документирующий комментарий */`
  - Первое предложение становится кратким описанием
  - Основной текст описывает функциональность
  - Специальные теги начинаются с @ (@param, @return)
  - HTML-теги — для форматирования

Теги для классов

ИТМО

- Базовые теги с основной информацией
  - @author — указывает автора класса
    - @author Anton Gavrilov
  - @version — версия класса
    - @version 1.2
  - @since — версия, когда элемент был добавлен
    - @since 1.1
  - @see — ссылка на связанный класс, интерфейс, метод
  - @deprecated — устаревший код с пояснением

Теги для методов

ИТМО

- Описывают параметры и возвращаемый тип
  - @param — описание параметра метода
    - @param *name description*
  - @return — описание возвращаемого значения
    - @return *description*
  - @throws — описание исключений
    - @throws *Exception description*

Теги для ссылок и форматирования

ИТМО

- Позволяют создать гипертекстовую разметку
  - {@link} — ссылка на другой класс или метод
    - `{@link MyClass#myMethod}`
  - {@code} — отображает текст как код
    - `{@code List<String>}`
  - {@literal} — игнорирует HTML-теги
    - `Use angle brackets {@literal < and >}`

Хорошая документация

ИТМО

- Документация — для пользователей кода (API)
- Описывать не "как?", а "что?" и "зачем?"
- Документация должна быть короткой и полезной
- Желательно описать все public и protected элементы
- Желательно описать предусловия и постусловия
- Иногда полезны примеры использования кода
- При изменении кода документация должна обновляться
- Лучше использовать английский язык

Документация для пакета

- package-info.java — файл для документирования пакета
- Содержит JavaDoc-комментарий
- Описывает пакет в целом и основные классы

Генерация документации

ИТМО

- Утилита javadoc
  - `javadoc -d doc *.java`
  - опции -author и -version

- Инструменты сборки
  - mvn javadoc:javadoc
  - gradle javadoc

- Генерируется индекс, списки и иерархия классов
- Можно настроить стили CSS

Плюсы JavaDoc

ИТМО

- Javadoc — стандарт для документирования Java-кода
  - Обязателен для публичных библиотек на Maven Central
  - Используется для всей стандартной библиотеки
  - Используется средствами статического анализа и ИИ-агентами
  - IDE отображает документацию во всплывающих подсказках

Тестирование и отладка

- Тестирование — процесс проверки программного обеспечения для выявления дефектов и оценки его соответствия требованиям
- Отладка — процесс поиска, анализа и исправления дефектов в программном коде

Основные базовые инструменты

- Использование assert
- Модульное тестирование
- Логирование
- Режим отладки в IDE и утилиты командной строки

Утверждения - assert

ИТМО

- Инварианты, которые **всегда** должны быть истинными
  - `assert x == 0 : "X should be zero";`
- Обычно используются на этапе разработки
- Если утверждение ложно, выбрасывается AssertionError
- По умолчанию отключены
- Включаются опцией -ea (-enableassertions)
- Не используются для валидации данных

Unit-тестирование

ИТМО

- Утверждения проверяют внутри приложения
- Unit-тесты — это код, который проверяет код
- Тест проверяет одну единицу (unit) кода — обычно метод
- Этапы теста (AAA)
  - Arrange (подготовка)
  - Act (выполнение)
  - Assert (проверка).

Фреймворк тестирования JUnit

- Тест — это Java-класс в каталоге src/test/java
- Тестовый метод помечается аннотацией @Test
- Зеленый = тест прошел, красный = есть ошибки

Пример

```
public class Calculator {
  public int add(int a, int b) {
    return a + b;
  }
}
```

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class CalculatorTest {
  @Test void testAdd() {
    // Arrange
    Calculator calculator = new Calculator();

    // Act
    int result = calculator.add(2, 3);

    // Assert
    assertEquals(5, result, "Wrong add result!");
  }
}
```

Методы Assertions в JUnit

ИТМО

- assertTrue() / assertFalse()
- assertNull() / assertNotNull()
- assertEquals()
- assertArrayEquals()
- assertThrows() — проверка выбрасывания исключения

Хорошие тесты

ИТМО

- FIRST
  - Fast (быстрые)
  - Isolated (изолированные)
  - Repeatable (повторяемые)
  - Self-Validating (самопроверяемые)
  - Timely (своевременные)
- Понятные имена тестовых методов
- Позитивные и негативные сценарии
- Независимость тестов друг от друга и от их порядка

Когда писать тесты

- До или вместе с написание кода
- TDD — Test Driven Development
  - Сначала пишем тесты без кода
  - Тесты не проходят
  - Пишем код, чтобы тесты прошли

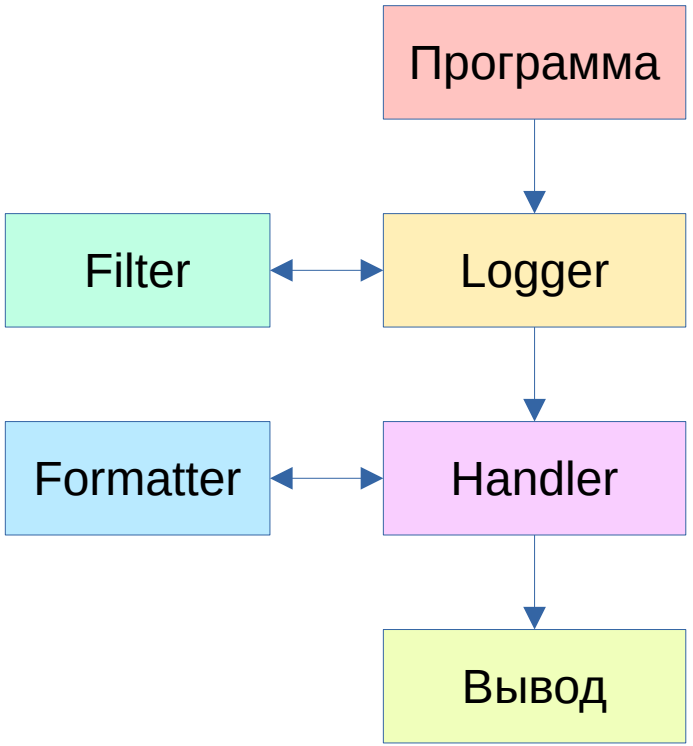
Логирование

ИТМО

- Позволяет получать сведения о работе программы
- Основное назначение — диагностика проблем
- Разные реализации
  - System.err.println (так плохо, но хотя бы не out)
  - java.util.logging
  - log4j / log4j2 / Logback и другие

Как работает логирование

- Logger — отправляет сообщения
- Level — уровень проблемы
  - CRITICAL, ERROR, WARNING,
  - INFO, CONFIG, FINE, ALL, OFF
- Handler — управляет выводом
- Filter — фильтрует ненужное
- Formatter — форматирует вывод



Пример

ИТМО

```
package mypackage;
import java.util.logging.Logger;
public class MyClass {
  private static Logger logger = Logger.getLogger(MyClass.class.getName());

  public void myMethod(int x) {
    if (x < 0) {
      logger.warning("X is negative");
    }
  }
}
```

```
Handler fh = new FileHandler("myclass.log");
Logger.getLogger("mypackage.MyClass").addHandler(fh);
Logger.getLogger("mypackage.MyClass").setLevel(Level.INFO);
```

Режим отладки в IDE

- Основные возможности
  - точки останова
  - пошаговое исполнение кода
  - просмотр значений переменных
  - отслеживание условий

Система контроля версий: зачем Git?

ИТМО

- VCS — система управления изменениями в коде
  - Хранение истории изменений
  - Сравнение версий
  - Простой возврат к любой версии
  - Совместная работа над проектом
- Git — распространенная популярная VCS
- GitHub — популярный хостинг для Git-репозиториев

Базовые функции и команды Git

ИТМО

- git clone *url* — скопировать репозиторий себе
- git init — создать новый репозиторий
- git add *file/dir* — добавить в область сохранения
- git commit -m *"сообщение"* — зафиксировать версию
- git push — отправить рабочий коммит в репозиторий

Дополнительные функции и команды

- git status — посмотреть состояние рабочей области
- git log — история коммитов
- git branch *name* — создание отдельной ветки
- git switch *name* — переключение на ветку
- git merge *name* — слияние веток

Как лучше пользоваться

- Частые и небольшие коммиты
  - Один коммит — одно изменение
- Тестирование и проверка кода перед коммитом
- Осмысленные сообщения коммитов
- Ветки под отдельные функции
- Не хранить пароли и персональные данные (.gitiignore)

Вопросы?