HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

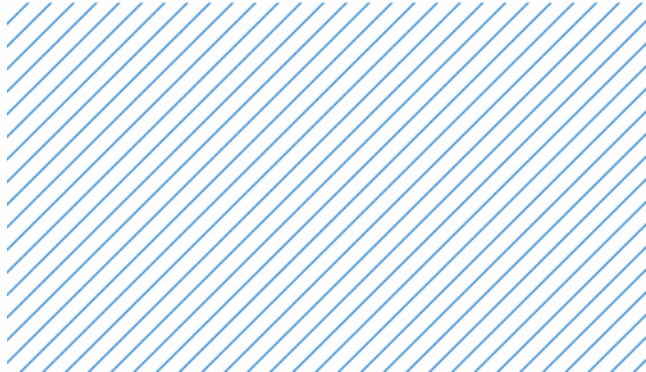
Лекция 6

Командная строка, компилятор Java и документация

Программирование на языке Java

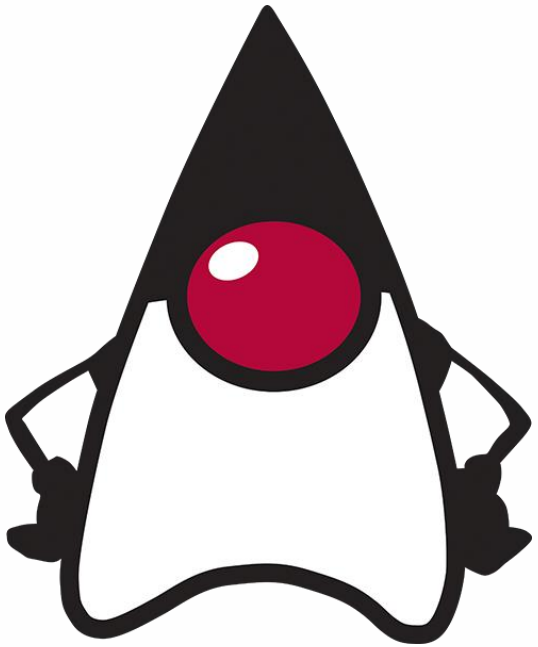
Роман Гуров

ВШЭ БИ 2021



Что такое JDK?

Java **D**evelopment **K**it – буквально набор разработки на Java – комплект программ, позволяющий создавать Java-приложения.



Существует несколько поставщиков JDK, отличающихся стабильностью, качеством поддержки и, иногда, функциональностью

Самый эталонный (и дорогой для коммерческого использования) – Oracle JDK

От тех же Oracle существует и открытая бесплатная версия – OpenJDK

Попробуем самостоятельно установить себе JDK…

Если где-либо вдруг не находит установленный JDK – проверьте переменные среды PATH и JAVA_HOME (не относится к IntelliJ IDEA)

Компилятор Java

Для компиляции кода используется утилита javac

Достаточно просто вызвать её, передав в командной строке имена исходных файлов:
`javac Main.java NotMain.java`

Результатом компиляции будут являться файлы с расширением `.class`:
`Main.class, NotMain.class, ...`

Скомпилированная программа хранится в этих файлах в формате *байт-кода*

Эти файлы можно «*прочитать*» утилитой javap

Запускатор Java

Файлы .class по сути и являются готовым скомпилированным java-приложением
Каждый из них по отдельности задаёт кусочек, соответствующий своему файлу с кодом

В отличие от нативных (например, на Windows это .exe) приложений, приложения Java не воспринимаются процессором напрямую

Для запуска Java-программы нужно использовать утилиту java

Использование: `java <имя класса, содержащего точку входа>`
Пример: `java Main`

А как по этому имени класса ищется конкретный файл?

Во-первых, в Java имя файла обязано соответствовать имени класса внутри него

Во-вторых, есть специальный параметр `-classpath`, позволяющий указать пути для поиска классов

Если этот параметр не указан, то поиск производится в директории, из которой вызвана утилита

JVM

А почему, собственно, нельзя запускать приложение напрямую?
И кто же их в итоге запускает?

Java **V**irtual **M**achine – виртуальная машина Java – специальная программа,
читающая и исполняющая байт-код Java

По сути, ваше приложение запускается не напрямую, а через посредника – JVM

Самое большое преимущество этого – кроссплатформенность
Байт-код Java не привязан ни к какой операционной системе или архитектуре процессора

Если есть реализация JVM для некоторой платформы, то все Java-программы уже работают на ней!

Также, использование виртуальной машины позволяет языку иметь более гибкую функциональность

Но всё это ценой меньшей производительности

JVM не только для Java

Байт-код Java не привязан к Java, никто не запрещает создать свой язык, который бы компилировался в байт-код Java, или даже написать такой компилятор для уже существующего

Примеры других языков на JVM:

- Clojure
- Scala
- Jython – имплементация языка Python
- jRuby – интерпретатор языка Ruby
- Kotlin – «убийца» Java от компании JetBrains
- Nashorn – движок JavaScript
- Groovy

Общая платформа позволяет всем этим языкам пользоваться богатым набор существующих Java-библиотек

JRE

Существует некоторый недостаток:
Как передать своё приложение пользователям, чтобы они смогли его запустить?

Не хотелось бы давать им JDK вместе со всеми компиляторами и вспомогательными утилитами

К сожалению, JVM не входит в поставку большинства операционных систем,
поэтому поставлять что-то придётся

Java **R**untime **E**nvironment – минимальная среда для выполнения программ

JRE уже входит в состав JDK

Если требуется только запускать приложения, но не заниматься их разработкой,
то JRE – корректный вариант

Можно поставлять конкретную версию JRE вместе со своим приложением,
или предложить пользователю загрузить и установить его самостоятельно с сайта Oracle

Jar-файлы

Приложение, разбросанное по куче .class файлов выглядит неудобно и не самодостаточно

Java позволяет запаковать всё нужное в один архив в формате .jar

Помимо классов, в jar-архиве хранится манифест –
файл, описывающий дополнительные сведения о программе

Например, в манифесте может быть указана точка входа

Технически, jar просто является zip-архивом и открывается любым архиватором

Jar-файлы

Для работы с jar используется утилита jar

Создание: `jar cfe MyProgram.jar Main <список .class файлов>`

Просмотр списка файлов: `jar tf MyProgram.jar`

Распаковка: `jar xf MyProgram.jar`

Запуск: `java -jar MyProgram.jar`

Запуск у явным указанием точки входа: `java -classpath MyProgram.jar Main`

Для последнего не требуется наличие точки входа в манифесте, поэтому jar можно создать вот так:
`jar cf MyProgram.jar <список .class файлов>`

Сторонние библиотеки

Как упоминалось ранее, в параметр `-classpath` можно указать сразу несколько путей

Применяется это для того, чтобы программа могла помимо стандартной библиотеки Java использовать и сторонние библиотеки

Если в нашей программе импортируется какой-то сторонний класс,
то он будет искаться в остальных перечисленных путях (и jar-файлах)

Попробуем воспользоваться сторонней библиотекой commons-lang…

Javadoc

Можно заметить, что все документации библиотек Java (и стандартной тоже) выглядят одинаково

Причина этому – удобная утилита javadoc, которая умеет генерировать документацию в формате HTML прямо из кода программы

Для добавления документации к функции внутри кода,
нужно использовать специальный комментарий /**

```java
/**
 * Печатает число ДВАЖДЫ!
 * <p>
 * Идеальная функция на случай, когда вас стабильно не понимают с первого раза.
 * Используется прямо из метода {@link #main}.
 *
 * @param n то самое число, которое надо напечатать дважды
 * @see #main
 */
public static void printNumberTwice(int n) {
    System.out.println(n);
    System.out.println(n);
}
```

Посмотрим на это
в действии…

JIT-компиляция

Можно подумать, что JVM постоянно работает в режиме интерпретатора:
читает байткод и выполняет соответствующий ему машинный код

Но постоянно иметь такую прослойку – очень медленно, особенно для арифметики

Сложение чисел – это одна быстрая инструкция процессора, но если для её исполнения нужно каждый раз смотреть байткод и выбирать соответствующее действие,
то время выполнения может вырасти в несколько раз

Чтобы избавиться от этой проблемы, применяется **J**ust-**i**n-**T**ime компиляция – байт-код может превращаться в машинный код на лету, во время исполнения программы

Hotspot – имплементация JVM, которая умеет анализировать нагруженные участки кода (методы и циклы) и автоматически решать, когда выгодно применять JIT-компиляцию

Такая техника, когда динамическая перекомпиляция кода происходит на основании текущей статистики исполнения, называется *адаптивной оптимизацией*