нетология

ВВЕДЕНИЕ В JAVA: JDK, JRE, JVM, INTELLIJ IDEA

ВАСИЛИЙ ДОРОХИН

ВАСИЛИЙ ДОРОХИН

QuadCode, QA Engineer

Василий Дорохин

План занятия

1. Автоматизация
2. Введение в программирование
3. Языки программирования
4. Java
5. JDK, JRE, JVM
6. Работа в терминале
7. IntelliJ IDEA
8. Итоги

АВТОМАТИЗАЦИЯ

АВТОМАТИЗАЦИЯ

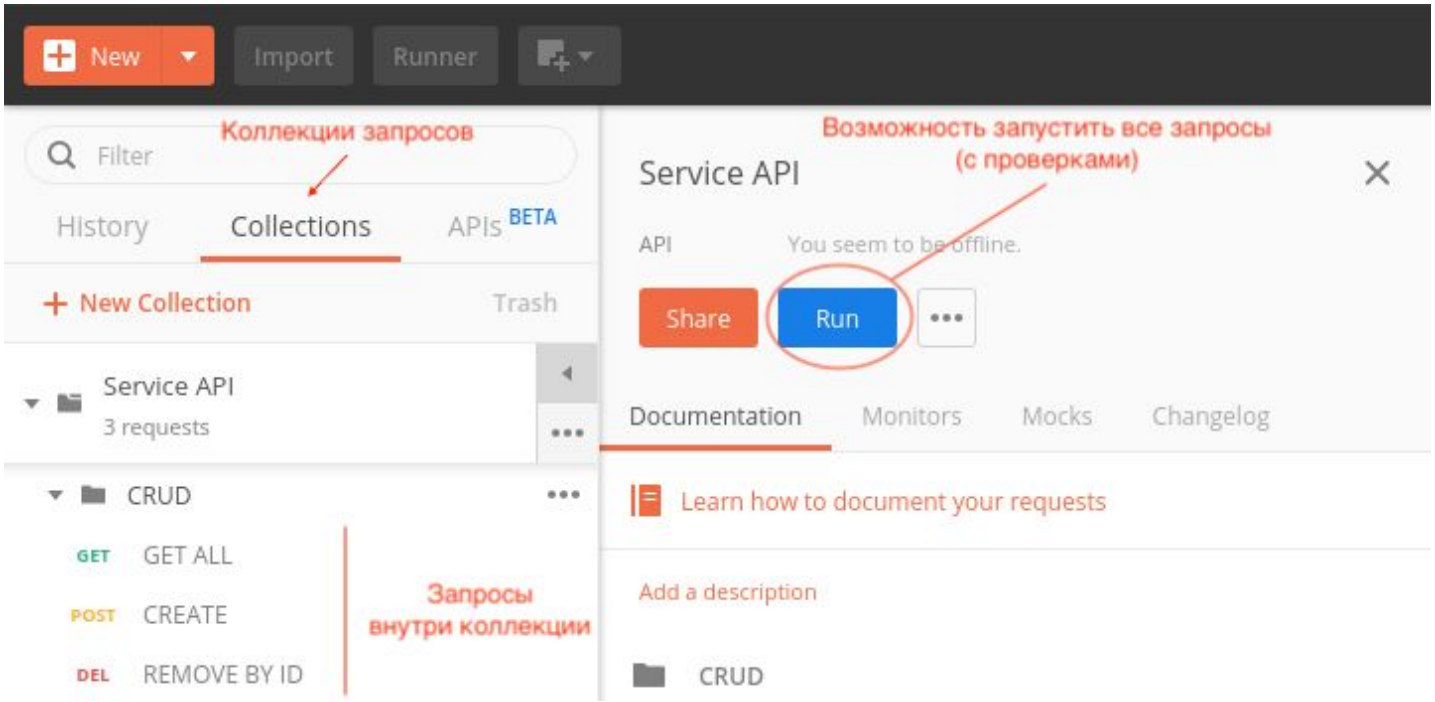
Из курса ручного тестирования вы знаете, что часть вашей работы — **это повторяющиеся задачи**. Например, проверка процесса авторизации на сервисе. Такие задачи важны и их нельзя пропускать, так как они входят в «критический путь».

Нет ничего страшного в том, чтобы проделать такие задачи несколько раз, но представьте, если вам придется выполнять их каждую неделю? Вы **быстро потеряете концентрацию внимания**, станете пропускать ошибки и в конечном счете можете устать от своей работы.

Для избежания таких последствий и существует автоматизация. **Автоматизация повторяющихся задач** позволит вам отдать однообразные задачи компьютеру, а самим заняться более творческими задачами.

АВТОМАТИЗАЦИЯ

Вы уже знакомы с инструментами, которые позволяют реализовать механизм «Record & Replay», например, Postman позволяет вам сохранить коллекцию HTTP-запросов и запускать их (вместо того, чтобы заново их вручную создавать).



АВТОМАТИЗАЦИЯ

Ключевое: разработчики инструментов не могут заложить всю функциональность в графические элементы управления (кнопки меню) — поэтому большинство предоставляют **возможность их «программировать»**. А именно — описывать нам собственную логику, как мы делаем в тест-кейсах.

Пример части тест-кейса перевода денег с карты на карту:

```
Ожидаемый результат:
- сумма на первой карте уменьшилась на 10 000 рублей
- сумма на второй карте увеличилась на 10 000 рублей
```

Вот только делать это надо в формате, который «понимает» компьютер, т.к. трактовка естественных языков (например, русского) — нетривиальная задача.

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

Программирование — процесс создания программного обеспечения.

Сложно и не понятно. Для нас программирование будет заключаться в том, чтобы с помощью специальных инструкций заставить «компьютер» выполнять нужную нам последовательность действий.

Например, мы хотим «запрограммировать» автоматический прогон тест-кейсов с индикацией `pass`/`fail`.

ЗАЧЕМ ОНО ТЕСТИРОВЩИКУ?

Q: Разве это не задача программиста — «программировать»?

A: Сейчас нет такого «задача программиста». Вся команда работает над одной задачей — создавать лучший продукт/сервис. Происходит взаимопроникновение и взаимопересечение обязанностей.

Чем большим количеством навыков вы обладаете (сюда относится и программирование), тем бóльший вклад вы можете внести в продукт.

ПРИЧИНЫ

Помимо возможности автоматизации есть ещё две причины, по которым «программирование» нужно:

1. **Понять, как мыслят программисты и где они ошибаются.** Будете знать, какие ошибки программисты допускают, в каких местах и почему. Это позволит вам лучше проектировать свои тесты.
2. **Понять, как работают программы и где могут возникнуть проблемы.** Если вы не знаете СУБД и как программы «общаются» с ними, вы не сможете предположить, какие там могут быть проблемы.

ПРОГРАММИРОВАНИЕ

Для нас **программирование** — написание инструкций, заставляющих «компьютер» выполнять нужную нам последовательность действий.

Осталось понять, **как писать эти инструкции и как их «запускать» на исполнение**.

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Выглядят эти инструкции примерно следующим образом:

```
...
00000000 00000000 00000000 00000000 00000000 00000000
01101000 01100101 01101100 01101100 01101111 00100000
01110000 01110010 01101111 01100111 01110010 01100001
01101101 01101101 01101001 01101110 01100111 00100001
01011100 01101110 00000000 00000000 00000000 00000000
...
00000000 00000000 00000000 00101110 01100100 01100001
01110100 01100001 00000000 00101110 01100010 01110011
01110011 00000000 00101110 01110100 01100101 01111000
01110100 00000000 00101110 01110011 01101000 01110011
01110100 01110010 01110100 01100001 01100010 00000000
00101110 01110011 01111001 01101101 01110100 01100001
01100010 00000000 00101110 01110011 01110100 01110010
01110100 01100001 01100010 00000000 00101110 01110010
01100101 01101100 01100001 00101110 01110100 01100101
01111000 01110100 00000000 00000000 00000000 00000000
...
```

Это часть программы, печатающей на экран фразу hello programming!.

Понятно, что ни о какой понятности тут речи нет, и пытаться это воспроизвести по памяти — задача та ещё.

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Тот же текст, но уже с «подсказками»:

```
...
00000000 00000000 00000000 00000000 00000000 00000000  ......
01101000 01100101 01101100 01101100 01101111 00100000  hello
01110000 01110010 01101111 01100111 01110010 01100001  progra
01101101 01101101 01101001 01101110 01100111 00100001  mming!
01011100 01101110 00000000 00000000 00000000 00000000  \n....
...
00000000 00000000 00000000 00101110 01100100 01100001  ....da
01110100 01100001 00000000 00101110 01100010 01110011  ta..bs
01110011 00000000 00101110 01110100 01100101 01111000  s..tex
01110100 00000000 00101110 01110011 01101000 01110011  t..shs
01110100 01110010 01110100 01100001 01100010 00000000  trtab.
00101110 01110011 01111001 01101101 01110100 01100001  .symta
01100010 00000000 00101110 01110011 01110100 01110010  b..str
01110100 01100001 01100010 00000000 00101110 01110010  tab..r
01100101 01101100 01100001 00101110 01110100 01100101  ela.te
01111000 01110100 00000000 00000000 00000000 00000000  xt....
...
```

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Поскольку писать «так» неудобно — появились более высокоуровневые инструменты, позволяющие писать в более понятном нам формате.

Например, языки ассемблера:

```
section .data
      msg db "hello programming!”,0
section .bss
section .text
      global main
main:
      mov rax, 1;
      mov rdi, 1;
      mov rsi, msg;
      mov rdx, 18;
      syscall
      mov rax, 60;
      mov rdi, 0;
      syscall
```

Важно: процессор не понимает того, что здесь написано, этот текст нужно преобразовать в формат, понятный процессору.

ЯЗЫКИ ПРОГРАММИРОВАНИЯ

Например, «та же программа» (делающая то же самое), на Java:

```java
package ru.netology;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello programming!");
    }
}
```

На JavaScript:

```javascript
console.log("Hello programming!")
```

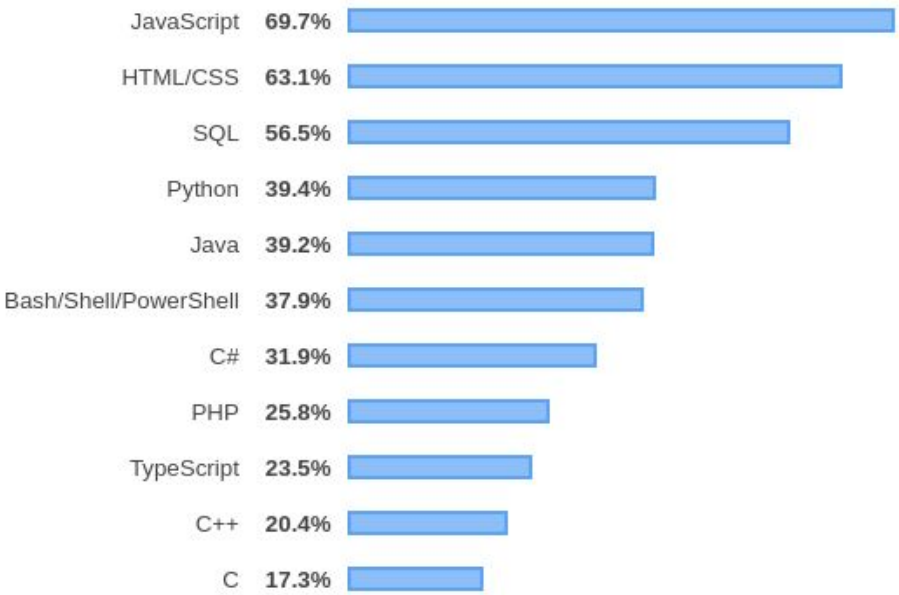
На Python:

```python
print("Hello programming!")
```

Какой же из языков выбрать?

STACKOVERFLOW

Можно взять статистику, например, StackOverflow* (популярность языков):



Важно: у любой статистики есть свои особенности, и популярность может не отражать востребованность или удобство.

В первой пятёрке HTML/CSS и SQL не являются языками общего назначения.

*StackOverflow — крупнейший портал формата Вопрос-Ответ в сфере ИТ и программирования в частности.

ВЫБОР ЯЗЫКА

Q: Что значит «общего назначения»?

A: Это значит, что есть возможность решать большинство задач (не является специализированным, как HTML — язык описания веб-страниц)

Q: Почему мы изучаем именно Java? Ведь самый популярный — JavaScript.

A: Мы опирались три ключевых аспекта:

1. Востребованность на рынке именно в сфере автоматизации тестирования.
2. Обучающие качества языка (логичность, зрелость, низкий уровень «магии«).
3. Удобство экосистемы — наличие инструментов, в том числе помогающих начинающим разработчикам.

Интересно, что в статистике JetBrains распределение выглядит по-другому*:

Какие языки программирования вы могли бы назвать основными в своей работе?
Выберите не более 3 языков.

Примечание*: нужно отметить, что и вопрос был задан другой.

ВЫБОР ЯЗЫКА

Q: Почему бы не учить сразу несколько самых популярных?

A: Вопрос времени и результатов: если вы направите усилия на изучение одного языка, прогресс будет гораздо бóльшим. Кроме того, после освоения одного языка, освоение других будет идти гораздо проще.

JAVA

САМЫЙ ВАЖНЫЙ СЛАЙД

В этом курсе мы будем проходить вещи, которые изобретались и совершенствовались годами.

Поэтому ключевое: **не пытайтесь разом понять все детали и вникнуть во всё**.

Многие вещи до вас будут «доходить» только спустя какое-то время, в этом нет ничего страшного, все через это проходили.

их целых два :)

САМЫЙ ВАЖНЫЙ СЛАЙД

Любое обучение происходит по спирали: вы сначала понимаете верхнеуровневые концепции и запоминаете некоторые вещи «как есть», нарабатываете навык, а уже затем разбираетесь в том, как это работает.

Воспринимайте всё как обучение вождению: вам показывают газ, тормоз и коробку передач. И учат для начала **просто трогаться с места и останавливаться**. И при этом вам инструктор не показывает схему внутреннего устройства этого всего — т.к. ваша задача освоить первый навык (буквально «запомнить»).

Только после того, как вы его освоите, можно разбираться с тем, как там всё устроено внутри.

JAVA

Java — язык общего назначения, используемый преимущественно в следующих сферах:

- корпоративные приложения;
- веб-сервисы;
- комплексные системы (обработка больших данных, системы очередей и т.д.);
- Android приложения*.

Примечание*: в сфере Android всё бóльшую популярность приобретает Kotlin.

О СОКРАЩЕНИЯХ И АББРЕВИАТУРАХ

В мире разработки программного обеспечения (далее — ПО) очень любят различные сокращения и аббревиатуры (а в мире Java их любят ещё больше).

Вам **обязательно** нужно их запомнить, т.к. без этого будет очень сложно. Начинается это «запоминание» с версий Java.

- 1.0 - 1996
- 1.1 - 1997
- 1.2 (J2SE 1.2) - 1998
- 1.3 (J2SE 1.3) - 2000
- 1.4 (J2SE 1.4) - 2002
- 1.5 (J2SE 5) - 2004
- 1.6 (Java SE 6) - 2006
- 1.7 (Java SE 7) - 2011
- 1.8 (Java SE 8) - 2014
- 9 (Java SE 9) - 2017
- 10 (Java SE 10) - 2018
- 11 (Java SE 11) - 2018
- 12 (Java SE 12) - 2019
- 13 (Java SE 13) - 2019
- Далее - каждые полгода (март и сентябрь)

LTS версии

ребрендинг, общее именование платформы: Java 2

изменение схемы именования - Java 5, Java 6 и т.д.
(внутренняя нумерация с 1.x сохраняется)

изменение схемы именования - Java 9, Java 10 и т.д.
(внутренняя нумерация с 1.x **не сохраняется**)

О СОКРАЩЕНИЯХ И АББРЕВИАТУРАХ

Q: Зачем мне это? Неужели нельзя взять просто самую последнюю версию?

A: К сожалению, нет по двум причинам:

1. В большинстве организаций используются Java 8 (осуществляется переход на Java 11).
2. Знание того, как называются версии и как формируются выпуски позволит вам понимать планы вашей команды по обновлению (и строить собственные планы по актуализации знаний).

Поэтому мы будем изучать 11 и говорить об особенностях 8 (затрагивая нововведения в Java 12+).

О СОКРАЩЕНИЯХ И АББРЕВИАТУРАХ

LTS (Long Time Support) — версия с длительной поддержкой
(Java 8 до 2019/2025, Java 11 до 2022/2026).

Следующей LTS версией будет Java 17, которая должна выйти в сентябре 2021.

Нужно запомнить, что до Java 9 есть два обозначения: 1.8 — это то же самое, что Java 8 (касается всех версий ниже 8).

Начиная с Java 9 никаких 1.9 уже нет.

SE, ME, EE

SE (Standard Edition) — платформа, предназначенная для разработки (программирования) приложений, работающих в серверных и десктоп окружениях.

ME (Micro Edition)* — платформа, предназначенная для разработки (программирования) приложений, работающих во встраиваемых и мобильных окружениях.

EE (Enterprise Edition)** — набор спецификаций, предназначенных для Enterprise*** приложений.

Примечание*: это не относится к Android.

Примечание**: в данный момент называется Jakarta EE.

Примечание***: корпоративное приложение, используемое крупными компаниями для автоматизации сложных бизнес-процессов.

SE, ME, EE

Q: Почему платформа?

A: Java — это не только язык программирования, но ещё и набор готового ПО для разработки и запуска приложений, написанных на языке Java (и не только), а также набор спецификаций.

Q: Какую же из трёх мы будем изучать?

A: Java SE.

Q: Как устроена платформа и как это всё вообще работает?

КОМПИЛЯЦИЯ VS ИНТЕРПРЕТАЦИЯ*

Чтобы понять, как всё устроено, рассмотрим упрощённую версию того, как программа, написанная на языке, который не понимает процессор, может быть исполнена им.

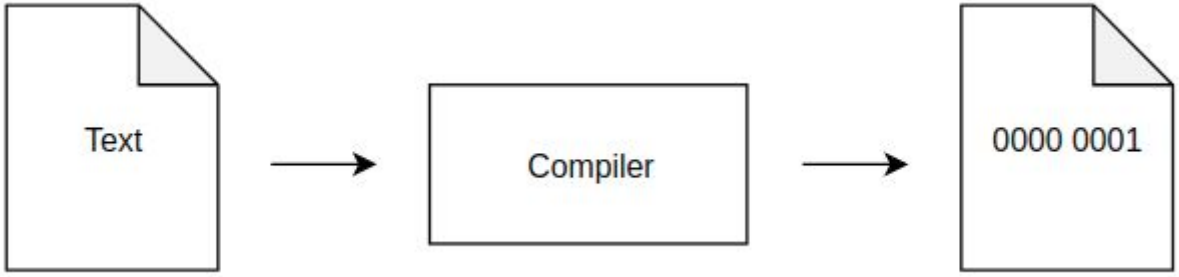
Есть два основных варианта:

1. Компиляция.
2. Интерпретация.

Примечание*: мы рассматриваем верхнеуровневую схему, опуская несущественные, на данный момент, детали.

КОМПИЛЯЦИЯ

С помощью специальной программы (компилятор) можно перевести текст из формы, понятной нам, в форму понятную компьютеру (набор 0 и 1):

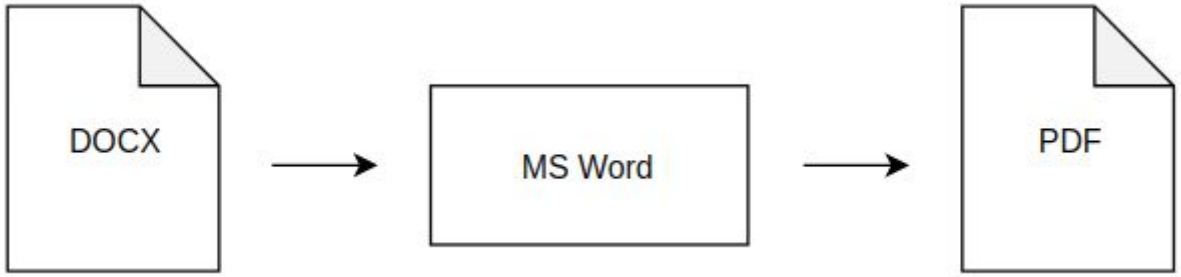


Например, для операционной системы Windows это может быть EXE-файл, который можно двойным кликом запустить на исполнение.

КОМПИЛЯЦИЯ

Q: Это как-то сложно...

A: Воспринимайте это следующим образом: если мы возьмём программу MS Word, то можем файлы формата DOCX преобразовать в файлы формата PDF:



Никаких принципиальных отличий нет — был файл одного формата, на базе него сгенерировали файл другого формата.

Так же, как и с MS Word, нас не интересует, как компилятор это делает, нас интересует только результат.

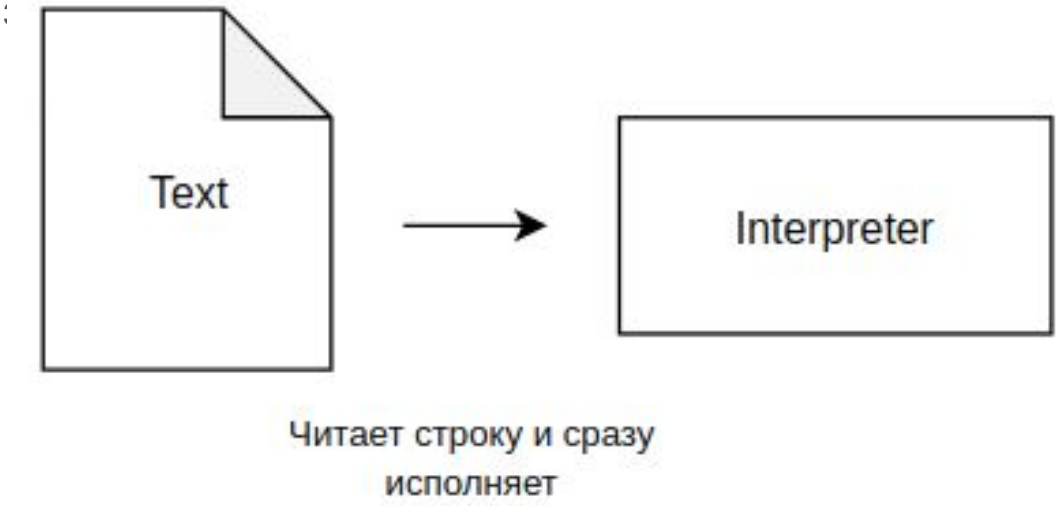
КОМПИЛЯЦИЯ

Важно: после того, как мы скомпилировали (сгенерировали) нужные файлы, программа, которая использовалась для генерации, больше не нужна.

Например, для просмотра PDF-файлов, MS Word уже не нужен, так же как для запуска EXE-файла не нужен будет компилятор.

ИНТЕРПРЕТАЦИЯ

С помощью специальной программы (интерпретатора) можно читать текст программы по строчкам и сразу исполнять его на процессоре. Т.е. интерпретатор, фактически, «компилирует» каждую строчку и сразу



ИНТЕРПРЕТАЦИЯ

Q: А есть аналог, как было с MS Word?

A: Да, упрощённо, например, плеер Youtube или Vk — загружает «кусочек» видео/аудио и сразу воспроизводит*.

Здесь нужно отметить: без интерпретатора (в отличие от компиляции) «воспроизвести» файл уже не удастся.

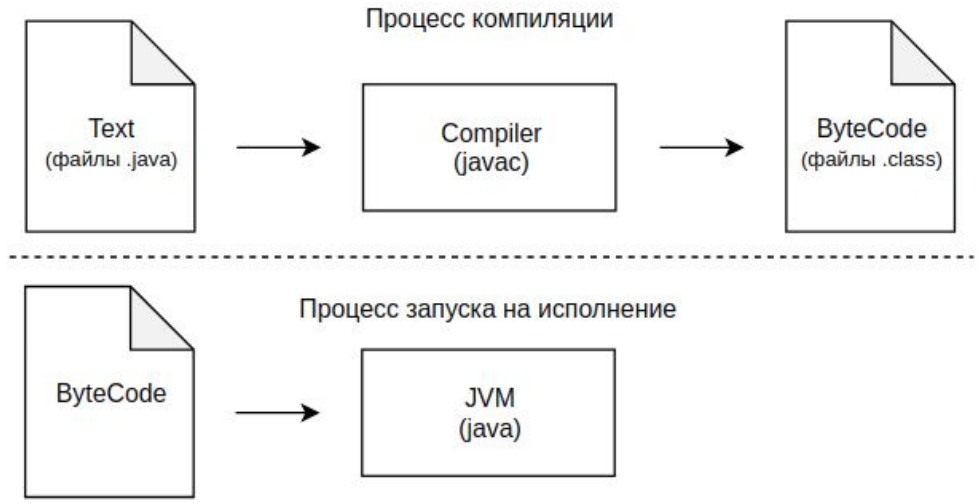
Q: Т.е. интерпретатор постоянно нужно будет «таскать» с собой?

A: Совершенно верно.

Примечание*: но программу нельзя воспроизвести «с середины«.

JAVA*

В Java используют гибридный подход: исходный код (текст программы) сначала компилируется в промежуточную форму с помощью компилятора, а затем исполняется с помощью JVM**:



Примечание*: мы рассматриваем верхнеуровневую схему, опуская несущественные, на данный момент, детали.

Примечание**: это не чистая «интерпретация«, а достаточно сложный процесс.

JAVA

Q: Почему сделали так?

A: Ответ на этот вопрос нельзя уместить в один слайд, но, как минимум, это позволяет обеспечить переносимость приложений.

Так, файлы EXE будут работать только в ОС Windows, т.к. в ОС Linux используется другой формат файлов.

В Java же, байткод будет исполняться на любой ОС для которой есть JVM* (и компилятор уже будет не нужен).

Q: Т.е. нам нужно скачать компилятор и JVM?

A: Почти верно.

JDK, JRE, JVM

JAVA

JVM (Java Virtual Machine) — программа, которая исполняет байт-код.
JRE (Java Runtime Environment) — JVM + стандартная библиотека (готовые, уже написанные для нас компоненты) + ряд инструментов.
JDK (Java Development Kit) — JRE + инструменты разработки (компилятор и другие).

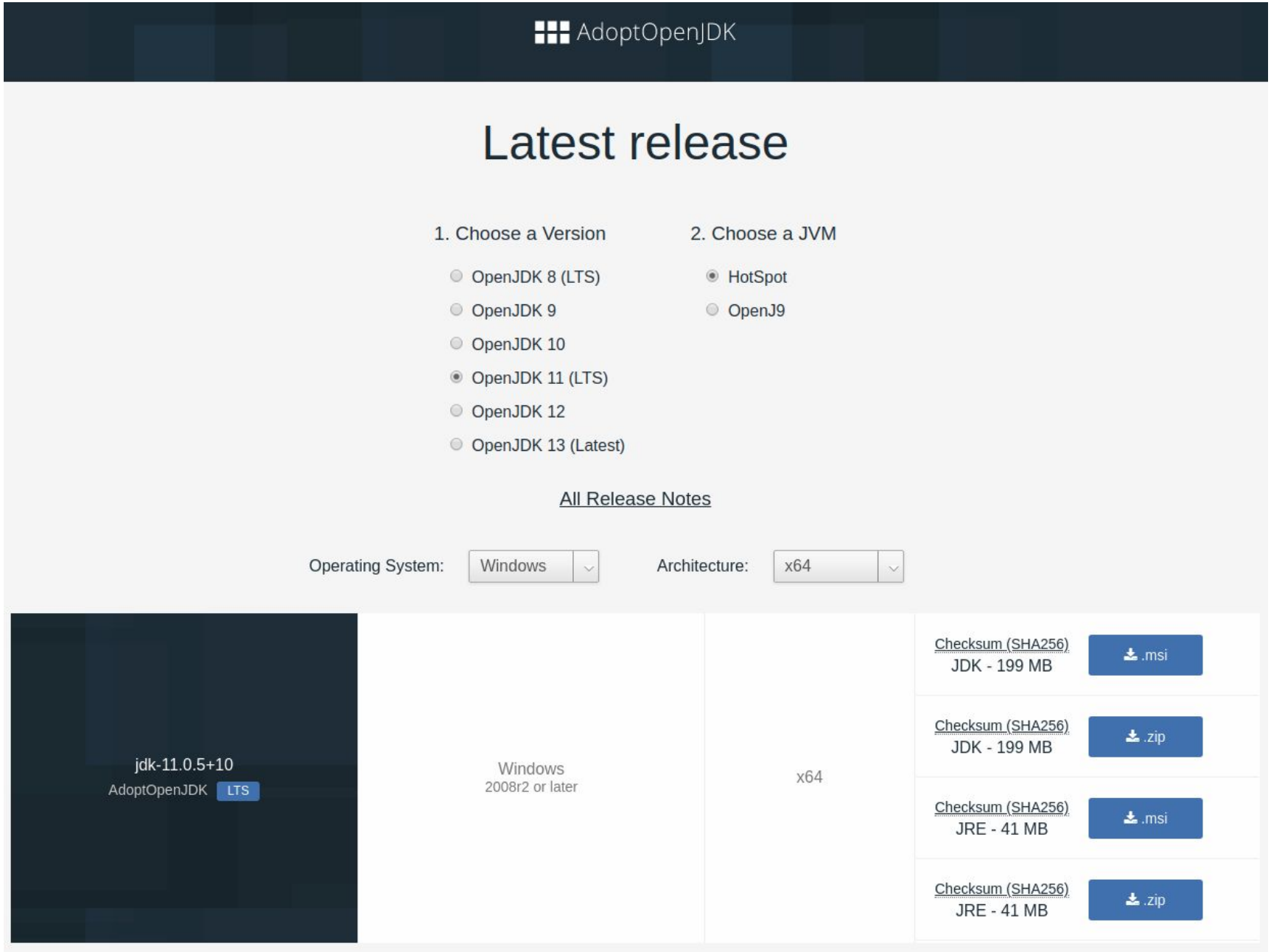
Таким образом, если мы хотим разрабатывать приложения на языке Java, нам нужен JDK.

OPENJDK

В мире Java существует несколько реализаций как JDK, так и непосредственно JVM.

Для обучения мы будем использовать референсную (эталонную) реализацию — OpenJDK и виртуальную машину HotSpot, которые доступны для свободного скачивания и установки.

Для нас специально подготовлены установочные файлы для самых популярных платформ: https://adoptopenjdk.net/releases.html.


AdoptOpenJDK
Latest release
1. Choose a Version
OpenJDK 8 (LTS)
OpenJDK 9
OpenJDK 10
OpenJDK 11 (LTS)
OpenJDK 12
OpenJDK 13 (Latest)
2. Choose a JVM
HotSpot
OpenJ9
All Release Notes
Operating System:
Windows
Architecture:
x64
jdk-11.0.5+10
AdoptOpenJDK
LTS
Windows
2008r2 or later
x64
Checksum (SHA256)
JDK - 199 MB
.msi
Checksum (SHA256)
JDK - 199 MB
.zip
Checksum (SHA256)
JRE - 41 MB
.msi
Checksum (SHA256)
JRE - 41 MB
.zip

УСТАНОВКА

Установка сводится к скачиванию установочного файла для нужной платформы (.msi — Windows, .pkg — для Mac OS, для Linux всё будет немного сложнее).

Вам предстоит самостоятельно в рамках ДЗ скачать и установить JDK: как нынешним уже тестировщикам и будущим автоматизаторам нужно привыкать по инструкциям и руководствам устанавливать и настраивать инструменты.

РАБОТА В ТЕРМИНАЛЕ

ОСНОВЫ РАБОТЫ В ТЕРМИНАЛЕ

Как тестировщикам и будущим автоматизаторам, вам достаточно много придётся работать в терминале.

Поэтому надеемся, что вы помните, как работать с ним по курсу Git.

Ключевое для нас:

- смена каталога осуществляется с помощью команды `cd <path>`, где путь может быть как абсолютным, так и относительным;
- переменная окружения `PATH` содержит список путей, в которых будет производиться поиск программ для запуска из терминала.

Установщики OpenJDK для Windows и Mac OS добавляют пути к Java в переменную `PATH` самостоятельно.

ПРОВЕРКА КОРРЕКТНОСТИ УСТАНОВКИ

```
worker@desktop MINGW64 ~
$ mkdir java-intro

worker@desktop MINGW64 ~
$ cd java-intro

worker@desktop MINGW64 ~/java-intro
$ pwd
/c/Users/worker/java-intro

worker@desktop MINGW64 ~/java-intro
$ java -version
openjdk version "11.0.5" 2019-10-15
OpenJDK Runtime Environment AdoptOpenJDK (build 11.0.5+10)
OpenJDK 64-Bit Server VM AdoptOpenJDK (build 11.0.5+10, mixed mode)

worker@desktop MINGW64 ~/java-intro
$ javac -version
javac 11.0.5
```

КОМПИЛЯЦИЯ И ЗАПУСК

Скопируем файл Main.java из репозитория к лекции в текущий каталог.

```
worker@desktop MINGW64 ~/java-intro
$ ls
Main.java

worker@desktop MINGW64 ~/java-intro
$ # компиляция

worker@desktop MINGW64 ~/java-intro
$ javac Main.java

worker@desktop MINGW64 ~/java-intro
$ ls
Main.class  Main.java

worker@desktop MINGW64 ~/java-intro
$ # запуск (расширение для .class-файла не указывается)

worker@desktop MINGW64 ~/java-intro
$ java Main
Hello programming
```

Символами # отмечены комментарии, не влияющие на работоспособность

КОМПИЛЯЦИЯ И ЗАПУСК

Q: Как-то не очень «продуктивно» оно выглядит... Это придётся делать каждый раз? И где набирать код, в nano или vim?

A: Мы продемонстрировали то, как «оно происходит». Конечно же, есть инструменты, которые будут помогать нам в этом.

IDE

IDE

IDE (Integrated Development Environment) — специальная среда разработки (приложение такое), предназначенное для оптимизации нашей работы в качестве программистов.

Обеспечивает:

- компиляцию, запуск на исполнение, отображение ошибок;
- автоподсказки, подсветку синтаксиса;
- генерацию кода(!) и многое другое.

А самое главное — на начальном этапе будут компенсировать ваше незнание языка и платформы.

Мастерство владения IDE будет во многом определять **вашу производительность и эффективность**, поэтому мы будем уделять этому большое значение.

INTELLIJ IDEA

Мы будем использовать самую популярную* среду разработки в мире Java:

Примечание:* по данным The State of Developer Ecosystem 2019.

INTELLIJ IDEA

Демонстрация набора и запуска «Hello programming!» в IntelliJ IDEA.

ИТОГИ

ИТОГИ

Давайте вспомним, как мы ответили сегодня на ключевой вопрос:

«Зачем тестировщику нужно программирование?»

ИТОГИ

Кроме того, мы обсудили достаточно много вопросов, касающихся Java:

1. Версии, LTS и текущую схему выпусков.
2. Редакции платформы Java SE, Java ME и набор спецификаций Java EE.
3. JDK, JRE, JVM и их взаимосвязь между собой.
4. Компиляцию и запуск тривиальной Java-программы.

ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задаем в чате Slack!
- Задачи можно сдавать по частям.
- Зачет по домашней работе проставляется после того, как приняты **все задачи**.

нетология

Задавайте вопросы и напишите отзыв о лекции!

ВАСИЛИЙ ДОРОХИН

Василий Дорохин