**Команды в консоли для компиляции и прочее**

javac Main.java //команда для компиляции java класса

javap -v Main.class // Инструмент java-ой виртуальной машины для просмотра байт кода

java Main //Запуск откомпилированного класса, с помощью указания главного класса. Здесь "Main"

*//Если класс находится в другой директории, то об этом нужно сообщить*

java –classpath classes\_dir Main //Вместо dir наша папка с файлом класса

**jar // архив с классами java. По своей сути это zip архив с доп. Информацией (мета инфа о программе), называемой манифест.**

*Например, имя программы, класса, версия,*

*В JDK инструмент для работы с такими классами* ***называется*** **jar**

Jar cfe hw.jar Main Main.class

// Пишем имя архива, имя класса и все файлы которые хотим упаковать в архив

jar tf hw.jar // Просмотр содержимого не распаковывая

jar xf hw.jar //Распаковка архива

*Если в jar прописан главный класс. То запустить программу очень просто:*

java –jar hw.jar //Запуск программы в архиве

*Если не прописан, но известен то:*

java –classpath hw.jar HelloWorld // В –classpath допускаются не только директории но и jar архивы

*Java программы часто зависят от сторонних библиотек классов, распространяемых как jar архивы, для того чтобы скомпилировать программу использующую сторонние классы, нужно объяснить компилятору где эти классы найти.*

Javac –classpath lib.jar Main.java //Добавляем сторонние классы в компиляцию

*Виртуальной машине тоже нужен доступ к библиотекам, поэтому:*

java – classpath lib.jar:hw.jar Main

*Под linux разделителем классов classpath будет* “:”, *а в Windows* “;”.

**Ввод и вывод в консоли**

**import** java.util.Scanner; // Для ввода с консоли нужно подключить

Scanner in = **new** Scanner(System.***in***); *// создаем экземпляр класса Сканер*

**int** num = in.nextInt(); //Используем эту функцию для ввода с консоли

*//Для разных типов, вводимых данных своя функция соответственно*

*//Например:*

**float** число = in.nextFloat();

**Языки появившиеся на основе JVM**

Clojur

Groovy

Scala

JRuby

Jython

**Как JVM будет исполнять байт код**

Простейший случай:

Последовательно анализирует инструкции, исполняет, и переходит к следующей.

Режим называется “**интерпретация”.**

**Интерпретатор** довольно просто написать, но к сожалению, будет работать медленно.

Первая версия java работала только в этом режиме.

Потом был добавлен *режим* ***just in time*** компиляции, или просто **JIT**.

В этом режиме VM сначала компилирует байт код, а затем он уже исполняется процессором.

Современный **JIT** способен на адаптивную компиляцию и оптимизацию байт кода в зависимости от того, сколько раз и с какими параметрами исполнился тот или иной участок кода.

Порой программы на JVM могут выполняться даже быстрей, благодаря тому, что hot spot(JVM) в своих оптимизациях использует знания о фактическом выполнении программы, что не доступно традиционным компиляторам.

**Среды разработки на JAVA**

IntellJ IDIA

NetBeans

Eclipse

**Быстрые команды в IDIA**

psvm - **public static void** main(String[] args) {}

sout - System.***out***.println();

CTRL+ALT+SPACE – Просмотр возможных функций в классах и т.п., при доступе через точку.

CTRL+ALT+SHIFT+L - Автовыравнивание

**Javadoc** — [генератор документации](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D0%B8) в [HTML](https://ru.wikipedia.org/wiki/HTML)-формате из комментариев исходного кода на [Java](https://ru.wikipedia.org/wiki/Java) от [Sun Microsystems](https://ru.wikipedia.org/wiki/Sun_Microsystems). Javadoc — стандарт для документирования [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) [Java](https://ru.wikipedia.org/wiki/Java). Большинство [сред разработки программного обеспечения](https://ru.wikipedia.org/wiki/%D0%A1%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F) автоматически генерируют [HTML](https://ru.wikipedia.org/wiki/HTML)-документацию, используя Javadoc.

**Комментарии:**

//Комментарий

/\*Комментарий\*/

/\*\*

\* специальные комментарии

\*

\*/ //*Те, что используются в javadoc*

**Правила названия классов Методов и Данных**

**Класс** – начинается с заглавной буквы, Несколько слов записываются слитно и каждая начинается с заглавной буквы.

Переменные и методы – именуются со строчной буквы.

**Сборка финальной программы из исходников**

Простейший случай сборки — это вызов компилятора.

В реальности это множество шагов, таких как:

* Скачивание библиотек,
* Генерация части исходников из каких ни будь метаданных,
* Компиляция,
* Прогон тестов,
* Упаковка собранных классов в jar архив,
* Генерация документации,
* И т.д.

Для автоматизации этих процессов есть системы сборки программ:

1. Ant – build.xml
2. Gradle – build.gradle
3. Maven – pom.xml

Если в файле проекта лежат gradle или maven то можно автоматически экспортировать проект в свою среду разработки.