Java

**Java**— строго типизированный объектно-ориентированный язык программирования общего назначения, разработанный компанией Sun Microsystems. Дата ввода в эксплуатацию – 23 мая 1995 г.

***Особенности языка Java***

1. **Язык Джава простой в сравнении с С++.**

- При создании Java взяли C++ и выкинули все лишнее. Потом вернули, правда, большую часть, но тем не менее. По сути, полностью построен на синтаксисе языке С и С++ и имеет не так много базовых команд (список?)– около полусотни. По сути, все такие же, за исключением const и goto, которых в Java нет. Спецификация описания языка – 60 страниц. (вообще, 800, но, наверное, тут речь о прям базовой спецификации).

- В Джава, в отличие от С, нет указателей, однако внутри они есть и механизм работы построен на указателях (например, говорящее название NullPointerException)

Указатель – переменная, значением которой является адрес ячейки памяти. То есть указатель ссылается на блок данных из области памяти, причём на самое его начало. Указатель может ссылаться на переменную или функцию. Для этого нужно знать адрес переменной или функции. Так вот, чтобы узнать адрес конкретной переменной в С++ существует унарная операция взятия адреса &. Такая операция извлекает адрес объявленных переменных, для того, чтобы его присвоить указателю.

Указатели используются для передачи по ссылке данных, что намного ускоряет процесс обработки этих данных (в том случае, если объём данных большой), так как их не надо копировать, как при передаче по значению, то есть, используя имя переменной. В основном указатели используются для организации динамического распределения памяти, например при объявлении массива, не надо будет его ограничивать в размере.

int x;

int \*y = &x; *// От любой переменной можно взять адрес при помощи операции взятия адреса "&". Эта операция возвращает указатель*

int z = \*y; *// Указатель можно разыменовать при помощи операции разыменовывания "\*". Это операция возвращает тот объект, на который указывает указатель*

Ссылки — это то же самое, что и указатели, но с другим синтаксисом и некоторыми другими важными отличиями. Следующий код ничем не отличается от предыдущего, за исключением того, что в нём фигурируют ссылки вместо указателей:

int x;

int &y = x;

int z = y;

Главное отличие ссылки от указателей в том, что указатель это целое число и поэтому для него доступны операции с целыми числами, а для ссылки доступны только операции копирования и разыменования.

- В Java есть Garbage Collector – по сути, автоматическое управление памятью (тк указателей нет, вручную управлять мы не можем). GC автоматически периодически удаляет объекты из памяти, на которые ничто не ссылается. Об этом дальше.

- В Java не используются заголовочные файлы и команды препроцессора.

В языках программирования Си и C++ заголовочный файл — файл, содержимое которого автоматически добавляется препроцессором в исходный текст в том месте, где располагается некоторая директива - #include <file.h> в Си. Это основной способ подключить к программе типы данных, структуры, прототипы функций, перечисляемые типы и макросы, используемые в другом модуле. По умолчанию используется расширение .h; иногда для заголовочных файлов языка C++ используют расширение .hpp.

Заголовочный файл в общем случае может содержать любые конструкции языка программирования, но на практике исполняемый код (за исключением inline-функций в C++) в заголовочные файлы не помещают. Например, идентификаторы, которые должны быть объявлены более чем в одном файле, удобно описать в заголовочном файле, а затем его подключать по мере надобности.

Препроцессор — это компьютерная программа, принимающая данные на входе и выдающая данные, предназначенные для входа другой программы (например, компилятора). Наиболее частый случай использования препроцессора — обработка исходного кода перед передачей его на следующий шаг компиляции.

Значит, **программа на Джава** – это директория (*package*), внутри которого могут находиться другие директории и файлы. Один файл в общем случае – это один класс. Названия файлов совпадают вплоть до регистра с названием класса. Эти файлы имеют расширение *.java*

1. **Java - объектно-ориентированный язык.**

- В нем работают 3 ключевых принципа ООП – инкапсуляция, наследование и полиморфизим.

- Однако, есть кое-что в Java, что делает его не 100 проц Объектно-ориентированным. Это примитивные типы данных. Без оберток они не являются объектами. То есть, они передаются не ссылками, а значениями. Обычные типы (в си их нет) добавлены для ускорения работы Джава на первых этапах ее развития. Они весят меньше (int весит 4 байта, когда пустая оболочка Integer, которая хранит ссылку – уже 8 байт + ее значение). А в классах – обертках реализуются методы + они с Дженериками работают

1. **Java – распределенный язык программирования.**

Он создавался в середине 90-х, когда был период развития интернета. Его изначально создавали для обеспечения логики клиент-серверных приложений + многопоточное программирование.

РАСПРЕДЕЛЁННОЕ ПРОГРАММ́ИРОВАНИЕ, способ обработки данных, при котором различные части программы выполняются на нескольких компьютерах, образующих параллельную вычислительную систему с возможностью наращивания производительности.

1. **Java может работать на разных платформах.**

- Написал раз – работает везде

- Для того, чтобы писать код на Java необходима *JDK*.

- Java является интерпретируемым языком,

Интерпрета́ция — построчный анализ, обработка и выполнение исходного кода программы или запроса (в отличие от компиляции, где весь текст программы, перед запуском, анализируется и транслируется в машинный или байт-код, без её выполнения)

-Однако, в java есть компилятор (*javac*).

Дело в том, что компилятор преобразует программу сначала в *байткод*, который передается в JVM и там уже посредством интерпретатора исполняется построчно. (а в полноценном интерпретаторе – 1с например - построчно исполняется код написанный непосредственно на языке программирования). А в си и си++ мы компилируем написанный код непосредственно в машинный код, который исполняется операционной системой компьютера. А в Java – компилируем в байт-код и потом виртуальная машина джава построчно выполняет его сама.

По сути, JVM – это интерпретатор. Но также в нем есть и транслятор (JIT) – это внутренний компилятор байт-кода в машинный код для более быстрого исполнения.

*Транслятор* – это программа или техническое средство, которое переводит код с одного языка на другой.

*Компилятор* – это вид транслятора, который переводит исходный код в машинный код.

Тк Java код исполняется Java машиной, а она работает с байткодом, то javac можно назвать компилятором. Правда, компилятором, можно назвать еще и потому, что компилятор, помимо трансляции исходного кода на машинный код, осуществляет еще и сборку исполняемой машинной программы.

Если компилятор генерирует исполняемую машинную программу на машинном языке, то такая программа непосредственно исполняется физической программируемой машиной (например компьютером). В других случаях исполняемая машинная программа выполняется соответствующей виртуальной машиной.

Виртуальная машина исполняет некоторый машинно-независимый код (например, байт-код, шитый код, p-код) или машинный код реального процессора. Помимо процессора, ВМ может эмулировать работу как отдельных компонентов аппаратного обеспечения, так и целого реального компьютера (включая BIOS, оперативную память, жёсткий диск и другие периферийные устройства). В последнем случае на ВМ, как и на реальный компьютер, можно устанавливать операционные системы (например, Windows можно запускать в виртуальной машине под Linux или наоборот). На одном компьютере может функционировать несколько виртуальных машин.

***Как скомпилировать файлы через консоль:***

- Через командную строку можно скомпилировать .java файл, если зайти через консоль в папку, в которой он находится и выполнить команду javac Test.java, где Test - имя файла.

- Чтобы запуситить один файл – нужно набрать java Test (где Test – имя файла)

Не нужно указывать .class

- Если файлы .java находятся в папке src , то скомпиленные файлы находятся в папке bin. Или out или target если они скомпилированы через idea .

\_\_\_

Чтобы несколько файлов скомпилить через консоль, нужно находясь в корневой папке проекта вызвать команду javac -d bin ./src/\*

***-d*** это флаг, после которого следует указать расположение, куда попадут скомпилированные классы. Это очень удобно, так как перекладывать, например, 1000 классов — очень трудоемкий процесс.

***bin*** - это название папки, в которую будут скомпилены файлы

**./src/\*** - здесь src – расположение файлов, которые надо скомпиллировать, а \* - команда для компиляции.

- Теперь файлы появились в папке bin и чтобы их запустить, надо набрать команду

java -classpath ./bin TestMain

где TestMain – это главный класс, в котором есть Main – через который запускать будем, .class писать тоже не нужно.

-***classpath*** - флаг, после которого следует указать местоположение скомпилированных классов. Java будет искать главный класс и все сопутствующие именно в этой директории.

- ***./bin*** название папки, в которой находятся скомпилированные классы.

\_\_\_

- Чтобы удобнее было работать с группой файлов, вместо многочисленных .class можно сделать один специальный архив - ***.jar*** файл

- для этого нужно в корневой папке проекта дополнительно создать файл manifest.mf

-В нем надо прописать:

main-class: src.TestMain

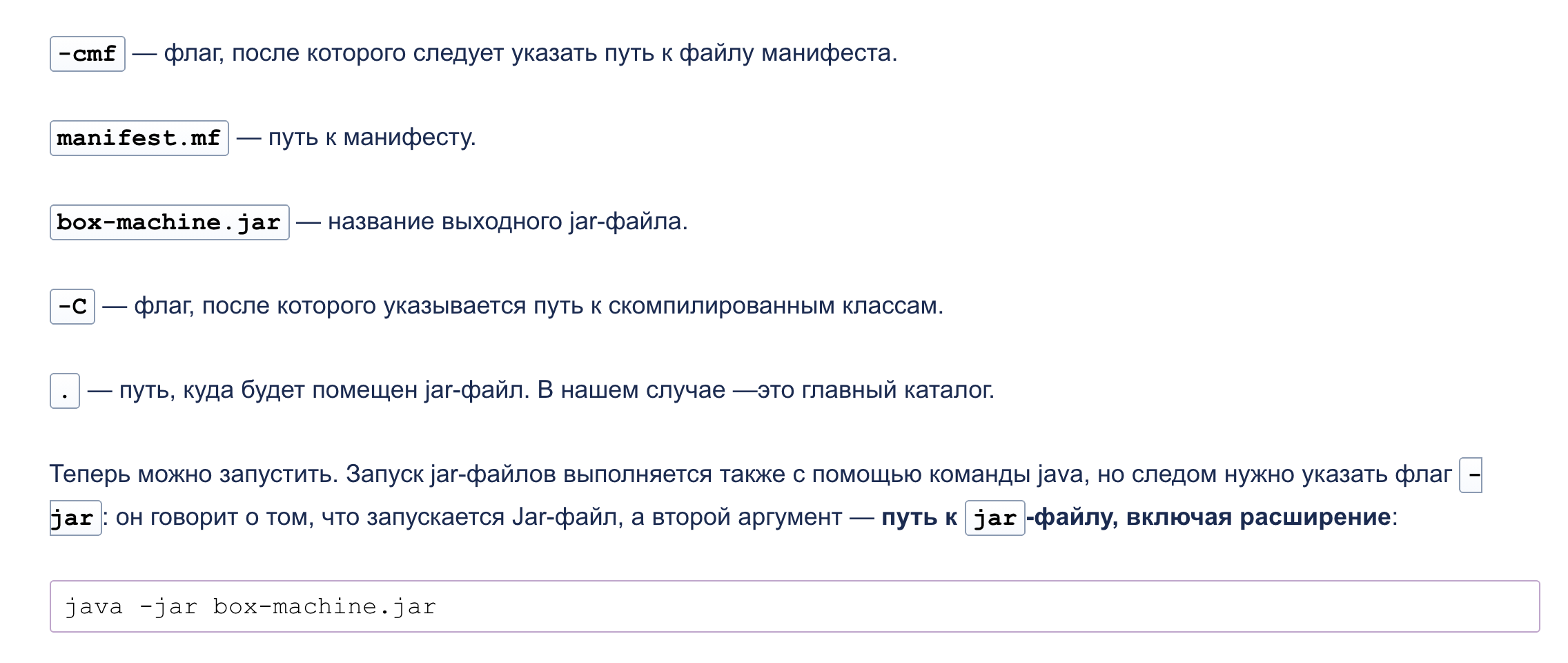
class-path: bin/

- соответственно, указываем главный класс (где main, но без .java)

а также указываем папку, в которую компилим джарку

jar -cmf manifest.mf box-machine.jar -C bin .

- это команда для создания джарки



Ну тут вместо box-machine.jar у нас будет Test.jar

+ Также можно компилить через idea или через idea+ сборщик (например, Maven)

***Загрузка .class или .jar в JVM***

***структура .class файла***

- После компиляции файл имеет текстовый вид байт-кода и содержит в себе информацию о приложении, расположенную в строгом порядке:

- Файл начинается со стартового (магического) числа: 0xCAFEBABE. Данное число присутствует в каждом классе и является обязательным флагом для JVM: с его помощью система понимает, что перед ней class-файл.

- Следующие четыре байта class-файла содержат старший и младший номера версий Java. Они идентифицируют версию формата конкретного class-файла и позволяют JVM проверять, возможна ли его поддержка и загрузка.

- С девятого байта идёт пул констант, в котором содержатся все константы нашего класса. Так как в каждом классе их может быть различное количество, то перед массивом находится переменная, указывающая на его длину, то есть пул констант представляет из себя массив переменной длины. Каждая константа занимает один элемент в массиве. Во всём class-файле константы указываются целочисленным индексом, который обозначает их положение в массиве. Начальная константа имеет индекс 1, вторая константа — 2 и т. д.

-Прочитав блок с константами, JVM переходит к следующим двум байтам — флагам доступа, которые определяют, описывает этот файл класс или интерфейс, общедоступный или абстрактный, является ли класс финальным.

-Имена класса и его родительского класса хранятся в массиве констант, на которые указывают последующие 4 байта в файле.

-Если это не класс а интерфейс, то **так** как класс может наследоваться от множества интерфейсов одновременно, то хранить необходимо массив ссылок на пул констант. То есть за определением класса и его родительского класса идёт число, характеризующее размер массива интерфейсов, и сам массив.

-Далее мы переходим к самому важному месту в любом классе — его методам, именно в них сосредоточена вся логика любой программы, весь исполняемый байт-код.

- Ситуация абсолютно аналогична описанным выше полям. В массиве переменной длины содержатся структуры, в которые входит полное описание сигнатуры метода: модификаторы доступа, имя метода и его атрибуты, которые также представляют из себя структуру, так как их может быть множество и каждый из них может принадлежать разным типам.

- В последнем блоке идёт дополнительная мета-информация, например имя файла, который был скомпилирован. Она может присутствовать, а может и нет. В случае каких-то проблем JVM просто игнорирует этот блок.

***Загрузка классов***

Весь процесс загрузки классов в JVM и иерархию загрузчиков пока описывать не буду – напишу лишь, что для каждого класса есть свой загрузчик, у них есть своя иерархия, и право загрузки обычно представляется корневому загрузчику, однако, они имеют право делегировать загрузку загрузчикам рангом пониже, среди которых могут быть и специальные собственные загрузчики. Также помимо классов загружается вспомогательный аппарат для JVM, такой как модули Java SE и JDK. Перед загрузкой загрузчик проверяет, не был ли загружен класс ранее, чтобы ускорить работу



1. **Язык Java – надежный.**

- В Java по умолчанию добавлен инструментарий обработки исключений и ошибок

- По-умолчанию есть библиотеки для написания thread-safe програм, а некоторые потокобезопасны по-умолчанию (напр String тк imutable) или спец типы данных или коллекции. Но не все, тк иначе было бы очень медленно.

-Инструментарий по безопасному управлению памятью. Например, при перекомпиляции данных подгружаемой внешней библиотеки в случае Си нужна перекомпиляция всей программы, тк изменился объем памяти. А в Java не надо, тк Java сама управляет памятью. В Java есть возможность менять классы программы, не останавливая работу программы.

- В Java можно использовать модули – объеднение packageй в более сложные конструкции. Это дает возможность дополнительно ограничивать и более тонко регулировать области видимости данных.

- В Java есть *intelSence* – система интеллектуального отслеживания синтаксиса и ошибок на уровне компилятора.

1. **Язык программирования Java - безопасный.**

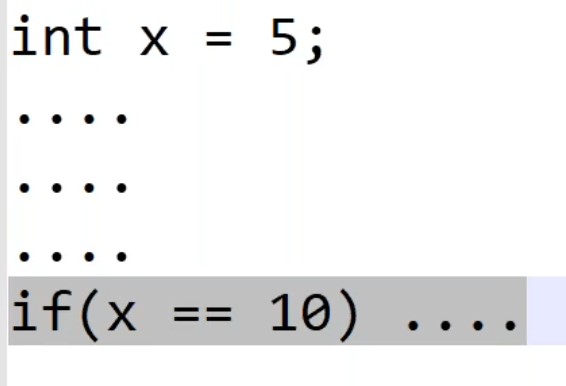
- В Java есть понятие native – это часть кода, которая написана на c и c++. Можно встраивать в джаву код на с и с++. Это очень удобно и быстро и круто. Но как – пока не знаю) Это Native-команды Java

Взломать написанную программу на Джава очень сложно

1. **Язык Java производительный.**

- На Java можно писать достаточно сложную математику. Это достигается за счет как раз использования primitive datatypes – можем быстро обрабатывать данные с операциями простой арифметики. (но си все равно быстрее – почему?). Чтобы определить реальную скорость работы программы на Java, нужно запустить ее раз 50 и взять средний результат последних 5 запусков. Это связано как раз с тем, что JVM исполняет код, и ее внутренняя логика не совсем линейна. Нужно много одинаковых запусков, прежде чем программа максимально оптимизируется. (например, при многопоточке)

- Ну и на Джава нужно аккуратно писать код – если тут volatile  не поставить то

 - при оптимизации эта строчка может быть убрана, а ведь x может меняться в др потоке

8. **Java – динамичный язык**

Можно писать код на одном устройстве, исполнять на другом.

Можно менять закрытую часть кода в момент исполнения программы (хз как, но можно)

Но нужно продумывать все заранее

\_\_\_\_\_

***на Java пишут:***

**beans** – это базовые классы в Java. Они создаются и используются для новых библиотек.

**консольные приложения** - самые простые приложения (вывод в консоль)

**GUI приложения** – оконные и с графическим интерфейсом – более сложные

**апплеты** – не используются сейчас тк не безопасны. Апплеты используются для предоставления интерактивных возможностей веб-приложений, которые не могут быть предоставлены HTML. Так как байт-код Java платформо-независим, то Java-апплеты могут выполняться с помощью плагинов браузерами многих платформ

**сервлеты**- Spring

**EJB** – для Java EE - устаревает

\_\_\_\_\_

***Основные package***:

java.lang – базовые команды

java.util – утилиты и вспомогательные конструкции

java.sql – библиотека для работы с бд

\_\_\_\_

***Refactoring*** – специальная интеллектуальная система переименовывания классов и пэкеджей

\_\_\_\_

В Java нет функций самостоятельных, там все функции привязаны к классам и называются ***методами***

\_\_\_\_\_

Макс точность в Java – 15 знаков после запятой

\_\_\_\_\_