1. **Собрать jar файл, находясь в другой директории:**

jar cfm Main.jar firstLab/MANIFEST.MF -C firstLab Main.class

1. **Создать и запустить jar без manifest**

java -cp Main.jar Main

1. **Модификаторы доступа**

* Private – методы доступны только в пределах класса
* Default (package visible) – доступно в пределах пакета (структура вложенных по какому-то признаку папок с размещенными в них классами (интерфейсами, перечислениями, аннотациями), необходимыми проекту)
* Protected – поля и методы доступны в пределах классов того же пакета и в классах-наследниках
* Public – поля и методы доступны везде

1. **Можно ли убрать public у начального класса?**

Да, класс станет default и будет доступен только в пределах своего пакета

1. **Статические методы, что ещё может быть статическим? (что вообще значит static)**

* Static метод – может вызываться без создания экземпляра класса
* Static переменная – относится к классу, а не к конкретным объектам
* Static вложенный класс - может быть создан без ссылки на экземпляр внешнего класса
* Final - неизменный

1. **Инициализаторы классов**

* Блоки кода, выполняемые при создании экземпляра класса. Бывают static и обычные
* Static инициализаторы - блоки кода, которые выполняются только один раз, когда класс загружается в память
* Не static инициализаторы выполняются же каждый раз.

1. **Что в каком порядке выполняется (инициализаторы статический и не статический, точка входа)**

Static инициализаторы – точка входа (метод Main) – не static инициализаторы – конструкторы класса

1. **в общих чертах про jdk / jre**

JDK – Java development kit. Комплект разработки ПО для Java.

Структура JDK:

* + JRE (Java Runtime Environment) минимальный набор инструментов, необходимых для выполнения Java-программ
    - JVM (Java Virtual Machine) интерпретирует байт-код
    - Стандартные библиотеки классов (например, java.lang, java.util, java.io и другие)
  + Компилятор (javac) преобразует исходный код Java (файлы .java) в байт-код (файлы .class), который затем выполняется JVM
  + Инструменты отладки (jdb) позволяет пошагово выполнять программы, устанавливать точки останова
  + Инструменты мониторинга (jvisualvm) графический интерфейс для анализа производительности
  + Архиватор (jar)
  + Документация (javadoc) Инструмент для автоматической генерации документации на основе комментариев

1. **Стек, кучи и области, на которые она разделена, metaspace**

* Стек – Область памяти. В нём хранятся локальные переменные, примитивные типы, указатели на объекты из кучи. Каждый поток имеет свой стек. При вызове функции создаётся новый фрейм стека.
* Куча – здесь динамически размещаются объекты и классы. Делится на eden, survivor space, old generation
  + Eden – здесь создаются все объекты. Когда эдем заполняется начинается сборка мусора и “живые” объекты переходят в survivor space
  + Survivor space – делится на s0 и s1 (меняются между собой, пока gc собирает мусор в одном, в другой помещаются объекты из эдема)
  + Old generation – Сюда попадают объекты, которые выдержали прошлые сборки мусора. GC заходит сюда редко, сборка мусора долгая
* Metaspace – здесь хранятся метаданные классов (например, инфа о загруженных классах, методах. данные о байт-коде методов, которые нужны JVM)

1. **Как GC (garbage collector) работает, откуда начинается обход**

Сборка мусора происходит для того, чтобы освободить память, которая больше не используется.

1. Когда программа создаёт новый объект (например, через оператор new), память для этого объекта выделяется в эдеме
2. Когда эдем заполняется, начинается Minor GC обход, удаляются все объекты без ссылок, остальные попадают в New generation
3. Далее Minor GC проходит по new generation и если объект выживает сколько-то (15) раз, то отправляется в Old generation, там сборка мусора реже и дольше
4. Если Old generation переполнено, то происходит Major GC, который очищает мусор во всей куче и metaspace
5. **Примитивные типы данных**

* Boolean
* Byte - 8
* Short - 16
* Int - 32
* Long - 64
* Char 0 - 2^16-1
* Float - 32
* Double - 64

1. **Ссылочные типы данных**

* Классы (String тоже здесь)
* Массивы
* Интерфейсы
* Классы-обёртки

1. **зачем нужны классы Int, Long и т.д. (можете про дженерики рассказать)**

Например, для работы с коллекциями по типу ArrayList и HashMap.

Для использования специальных методов и констант (проверить число NaN и тп)

Дженерики (механизм, позволяющий создавать классы и методы с типами, которые будут определены только во время использования) напрямую работают только с объектами

1. **как устроен многомерный массив**

В Java n-мерный массив представляет собой массив ссылок на другие (n-1)-мерные массивы. Одномерный массив хранится блоком, необязательно вместе с другими

1. **void**

Ключевое слово, обозначает, что метод ничего не возвращает.

1. **подробно разобрать что означает system out println**

System – встроенный класс

Out – static поле класса System. Означает выходной поток. Ещё есть in и err

Println – метод, выводит строку с переносом в конце.

* Когда мы вызываем System.out.println(""), Java ищет статическое поле out в классе System
* Поле out является объектом типа PrintStream, у которого вызывается метод println
* Метод выводит строку на консоль и завершает вывод новой строкой

1. **i++/++i, ++i + i++**

i++ сначала берём значение, потом увеличивает

++I сначала увеличивает, потом берёт

1. **NaN, inf и вообще немного про float/double (IEEE 754)**

* NaN – Not a Number получается, если операция не имеет смысла (0/0, корень логарифм отрицательного, 0^0, отрицательное в дробную степень)
* Infinity – получается при делении на 0 или другом выходе за диапазон у дробных чисел
* Float – 32 бита
* Double – 64 бита
* IEEE 754 – стандарт представления дробных чисел. value=(−1)sign×mantissa×2(exponent−смещение) число приводится в двоичную систему, нормализуется (1,XXXX + 2x), убирается 1 у мантиссы

1. **немного про math**

Класс java, входящий в основной пакет java.lang. Предоставляет набор методов для выполнений математических операций. В основном, если значение дробное, то возвращает double.

1. **Random**

Метод random в классе Math или

Класс Random генерирует псевдослучайные числа, в параметре использует seed для начального состояния генератора чисел. Если seed не задан, то по умолчанию он будет основан на текущем времени (в миллисекундах)

seed = (seed \* multiplier + addend) % modulus

1. **new**

Ключевое слово, выделяющее память под новый объект. Вызывает конструктор класса и возвращает ссылку на созданный объект в куче.

1. **байт-код**

Байт-код - промежуточное представление исходного кода после компиляции. Более удобен для JVM. javap посмотреть байт-код.

1. **jit компиляция. C1 C2**

Just in time компиляция превращает байт-код в **машинный код** на этапе исполнения программы. Если в программе присутствуют части кода, которые выполняются много раз, то их можно скомпилировать один раз в машинный код, чтобы в будущем ускорить их выполнение.

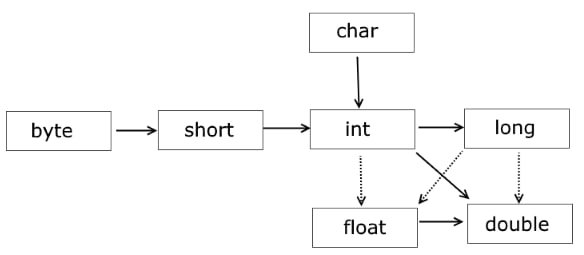
С1 – client компилирует код для первых 3 уровней сложности

C2 – server компилирует код для 4 уровня сложности

Чем сложнее уровень сложности, тем больше байт-кода будет закешировано в машинный код

1. **Приведение типов**

Маленький тип легко можно положить в большой



1. **Shebang не надо**

Специальная команда, которая пишется в начале скрипта и используется для указания интерпретатора, который должен быть использоваться для выполнения скрипта.

#!/usr/bin/env jshell