**Java Fundamentals**

1. JVM-JDK-JRE. Что это такое? Кто кого включает и как взаимодействуют.

**JVM** (Java Virtual Machine) исполняет байт-код Java, предварительно созданный из исходного текста Java-программы компилятором Java (javac). JVM обеспечивает платформо-независимый способ выполнения кода. Программисты могут писать код не задумываясь, как и где он будет выполняться.

**JRE** (Java Runtime Environment) - минимальная реализация виртуальной машины, необходимая для исполнения Java приложений, без компилятора и других средств разработки. Состоит из виртуальной машины и библиотек Java классов.

**JDK** (Java Development Kit) - комплект разработчика приложений на языке Java, включающий в себя компилятор, стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему JRE.

1. Как скомпилировать и запустить класс, используя консоль?

* Win + R
* cmd
* javac (name of class).java enter
* java (name of class) enter

1. Что такое classpath. Если в classpath есть две одинаковые библиотеки (или разные версии одной библиотеки), объект класса из какой библиотеки создастся?

Classpath – это параметр, который указывает приложениям где искать пользовательские классы. По этому адресу должны быть найдены все классы, для которых не применяются специальные [загрузчики](https://itsobes.ru/JavaSobes/kakie-sushchestvuiut-standartnye-zagruzchiki-klassov). На место поиска стандартных классов JRE этот параметр не влияет.

Есть два основных способа установки classpath: в переменной окружения ОС CLASSPATH, и в аргументе командной строки -cp (синоним -classpath). Второй способ предпочтительнее, потому что позволяет устанавливать разные значения для разных приложений. Значение по умолчанию – текущая директория.

Если есть две одинаковые библиотеки, то создастся объект класса той библиотеки, которую быстрее обнаружит.

1. Какие области памяти использует java для размещения простых типов, объектов, ссылок, констант, методов, пул строк и т.д.

Для оптимальной работы приложения JVM делит память на область стека (stack) и область кучи (heap). Всякий раз, когда мы объявляем новые переменные, создаем объекты или вызываем новый метод, JVM выделяет память для этих операций в стеке или в куче.

1. Стек (переменные, ссылки на объекты)

Стек работает по схеме LIFO (последним вошел, первым вышел). Всякий раз, когда вызывается новый метод, содержащий примитивные значения или ссылки на объекты, то на вершине стека под них выделяется блок памяти.

Когда метод завершает выполнение, блок памяти, отведенный для его нужд, очищается, и пространство становится доступным для следующего метода.

Основные особенности стека:

* Он заполняется и освобождается по мере вызова и завершения новых методов
* Переменные в стеке существуют до тех пор, пока выполняется метод в котором они были созданы
* Если память стека будет заполнена, Java бросит исключение java.lang.StackOverFlowError
* Доступ к этой области памяти осуществляется быстрее, чем к куче
* является потокобезопасным, поскольку для каждого потока создается свой отдельный стек

## 2. Куча (всё остальное)

Эта область памяти используется для объектов и классов. Новые объекты всегда создаются в куче, а ссылки на них хранятся в стеке.

Эти объекты имеют глобальный доступ и могут быть получены из любого места программы.

**Эта область памяти разбита на несколько более мелких частей, называемых поколениями:**

1. **Young Generation** — область где размещаются недавно созданные объекты. Когда она заполняется, происходит быстрая сборка мусора
2. **Old (Tenured) Generation** — здесь хранятся долгоживущие объекты. Когда объекты из Young Generation достигают определенного порога «возраста», они перемещаются в Old Generation
3. **Permanent Generation** — эта область содержит метаинформацию о классах и методах приложения, но начиная с Java 8 данная область памяти была упразднена.
4. Пакеты в java. Зачем применяются? Принцип именования пакетов.

Пакеты в java применяются для логической организации классов в наборы. Именуются с маленькой буквы.

цель.компания.проект.суть

1. Модификаторы доступа.

В Java используются следующие модификаторы доступа:

* **public**: публичный, общедоступный класс или член класса. Поля и методы, объявленные с модификатором public, видны другим классам из текущего пакета и из внешних пакетов.
* **private**: закрытый класс или член класса, противоположность модификатору public. Закрытый класс или член класса доступен только из кода в том же классе.
* **protected**: такой класс или член класса доступен из любого места в текущем классе или пакете или в производных классах, даже если они находятся в других пакетах
* **Модификатор по умолчанию**. Отсутствие модификатора у поля или метода класса предполагает применение к нему модификатора по умолчанию. Такие поля или методы видны всем классам в текущем пакете.

1. Почему метод main() объявлен как public static void?

Ключевое слово public -это модификатор доступа, который позволяет вызывать член вне класса.

static используется потому, что он позволяет вызывать main() без необходимости создавать экземпляр конкретного экземпляра этого класса.

void указывает, что main() не возвращает никакого значения.

main() должен иметь аргументы String как массивы, потому что JVM вызывает основной метод, передавая аргумент командной строки. Поскольку они хранятся в объекте строкового массива, он передается как аргумент main().

1. Что такое package level access? Пример использования.

Уровень доступа в пределах пакета. Его еще называют доступ по умолчанию. Такой уровень доступа устанавливается, если явным образом не задан спецификатор доступа (private, protected, public).

Например, есть два пакета (packageА и packageВ), в каждом пакете по два класса (classA1, classA2 and classB1, classB2). Из classA1 мы можем получить доступ к объектам classA2, но не можем получить доступ к объектам classB1, classB2. Аналогично для остальных классов каждого из пакетов. То есть из класса какого-либо пакета мы получаем доступ только к классу этого же пакета.

1. Может ли объект получить доступ к private-переменной класса? Если, да, то каким образом?

* внутри класса доступ к приватной переменной открыт без ограничений
* доступ вне класса осуществляется через геттеры и сеттеры
* если геттеры и сеттеры отсутствуют - рефлексия(крайне не рекомендуется).
* статический вложенный класс имеет полный доступ ко всем членам содержащего его класса, в том числе к членам, объявленным как private.

1. Классы-оболочки.

Классы-оболочки Java являются Объектным представлением восьми примитивных типов в Java. Все классы-оболочки в Java являются неизменными и final.



* Используйте классы-обертки, когда работаете с коллекциями.
* Используйте примитивные типы для того, чтобы ваши программы были максимально просты.

Еще одним важным моментом является то, что примитивные типы **не могут** быть **null**, а классы-оболочки — **могут**.

Также классы-оболочки могут быть использованы для достижения полиморфизма.

При помощи классов-оболочек можно работать с методами (toString(), parseXxx(), valueOf()).

1. Autoboxing и unboxing. Принцип действия на примерах.

Autoboxing:

Это автоматическая инкапсуляция примитивного типа в эквивалентную ему класс-обёртку всякий раз, когда требуется объект данного типа.

Autoboxing происходит:

1. При присвоении значения примитивного типа переменной соответствующего класса-обёртки.
2. При передаче примитивного типа в параметр метода, ожидающего соответствующий ему класс-обёртку.

Unboxing:

Это преобразование класса-обёртки в соответствующий ему примитивный тип. Если при распаковке класс-обёртка был равен **null**, произойдет исключение **java.lang.NullPointerException**.  
  
Unboxing происходит:

1. При присвоении экземпляра класса-обёртки переменной соответствующего примитивного типа.
2. В выражениях, в которых один или оба аргумента являются экземплярами классов-обёрток (кроме операции == и !=).
3. При передаче объекта класса-обёртки в метод, ожидающий соответствующий примитивный тип.
4. Вопрос:

int a = 1;

Integer b = 2;

int c = a+b; - результат? Выведет 3.

А каков будет результат если Integer b = null ? Не скомпилируется.

1. Что такое var? Достоинства и недостатки.

 Ключевое слово, которое избавляет нас от необходимости явно указывать тип локальной переменной.

Недостатки:

* риск снижения читабельности
* вероятность неправильного распознавания переменных (во избежание этого необходимо использовать литералы)
* **var не может быть использован, как тип возвращаемого значения или как тип аргумента метода**
* **лямбда выражениям и ссылкам на методы нужны явные типы**
* **тип var нельзя использовать в полях класса**
* **var нельзя использовать в блоке catch** (Однако, **var** отлично работает в блоке try-with-resources)

То есть самый большой минус – множество ограничений использования.

**Достоинства:**

* **удобно использовать при работе с потоками**
* **улучшает читабельность при замене типов большой длины (субъективно)**
* **способствует развитию языка**