INTERVIEW JAVA CORE JAVA DEVELOPER

Ответы на вопросы на собеседование Java core (часть 1).

Vasyl K 7:05:00 21 Комментарии

- **ЧЕМ ОТЛИЧАЕТСЯ JRE, JVM И JDK?**

JRE кратко – для работы. Java Runtime Environment (сокр. JRE) – минимальная реализация виртуальной машины, необходимая для исполнения Java-приложений, без компилятора и других средств разработки. Состоит из виртуальной машины – Java Virtual Machine и библиотеки Java-классов.
JDK кратко – для программирования. Java Development Kit (сокращенно JDK) – бесплатно распространяемый компанией Oracle Corporation (ранее Sun Microsystems) комплект разработчика приложений на языке Java, включающий в себя компилятор Java (javac), стандартные библиотеки классов Java, примеры, документацию, различные утилиты и исполнительную систему Java (JRE).
Java Virtual Machine (сокращенно Java VM, JVM) – виртуальная машина Java – основная часть исполняющей системы Java, так называемой Java Runtime Environment (JRE). Виртуальная машина Java интерпретирует Байт-код Java, предварительно созданный из исходного текста Java-программы компилятором Java (javac). JVM может также использоваться для выполнения программ, написанных на других языках программирования.

- **ОПИШИТЕ МОДИФИКАТОРЫ ДОСТУПА В JAVA.**

В Java существуют следующие модификаторы доступа:

- private: (используется конструкторах, внутренних классах, методах и полях класса) – Доступ разрешен только в текущем классе.
- default (package-private): (используется в классах, конструкторах, интерфейсах, внутренних классах, методах и полях класса) – Доступ на уровне пакета. Если класс будет так объявлен он будет доступен только внутри пакета.
- protected: (используется конструкторах, внутренних классах, методах и полях класса) Модификатор доступа на уровне пакета и в иерархии наследования.

- public: (используется в классах, конструкторах, интерфейсах, внутренних классах, методах и полях класса) – Модификатор доступа общественный, доступен всем.

Последовательность модификаторов по убыванию уровня закрытости: private, default ,protected, public).

- **ЧТО ТАКОЕ PACKAGE LEVEL ACCESS.**

Доступ из классов одного package-а в классы другого package-а.

- **ЧЕМ АБСТРАКТНЫЙ КЛАС ОТЛИЧАЕТСЯ ОТ ИНТЕРФЕЙСА? В КАКИХ СЛУЧАЯХ ВЫ БЫ ИСПОЛЬЗОВАЛИ АБСТРАКТНЫЙ КЛАСС, А В КАКИХ ИНТЕРФЕЙС?**

Абстрактный класс это класс, который помечен как "abstract", он может содержать абстрактные методы, а может их и не содержать.

Экземпляр абстрактного класса нельзя создать.

Класс, который наследуется от абстрактного класса может реализовывать абстрактные методы, а может и не реализовывать, тогда класс наследник должен быть тоже абстрактным. Также если класс наследник переопределяет реализованный в абстрактном классе родители метод, его можно переопределить с модификатором абстракт! Т.е отказаться от реализации. Соответственно данный класс должен быть также абстрактным также.

Что касается интерфейса, то в нем находятся только абстрактные методы и константы, так было до выхода Java 8. Начиная с Java 8 кроме абстрактных методов мы также можем использовать в интерфейсах стандартные методы (default methods) и статические методы (static methods).

- Default метод в интерфейсе – это метод в интерфейсе с по умолчанию реализованной логикой, который не требуется обязательно определять в реализации этого интерфейса.
- Static методы в интерфейсе – это по существу то же самое, что static-методы в абстрактном классе.

При реализации интерфейса, класс обязан реализовать все методы интерфейса. Иначе класс должен быть помечен как абстрактный. Интерфейс также может содержать внутренние классы. И не абстрактные методы в них.

Что же использовать Интерфейс или Абстрактный класс?

Абстрактный класс используется когда нам нужна какая-то реализация по умолчанию. Интерфейс используется когда классу нужно указать конкретное поведение. Часто интерфейс и абстрактный класс комбинируют, т.е. имплементируют интерфейс в абстрактном классе, чтоб указать поведение и реализацию по умолчанию. Это хорошо видно на примере свига:

```
1 class AbstractTableModel implements TableModel{
2 }
3 class MyTableModel extends AbstractTableModel{
4 }
```

Мы создаем свою модель таблицы с определенным поведением и уже с реализацией по умолчанию.

ВАЖНО! При реализации интерфейса, необходимо реализовать все его методы, иначе будет Fatal error, так же это можно избежать, присвоив слово abstract.

Пример:

```
interface I {
    public void F();
    public void say();
}

public abstract class A implements I {
    @Override
    public void say() {
        System.out.println("Hello!");
    }
    // public void F(); - not implemented
}
```

- **МОЖЕТ ЛИ ОБЪЕКТ ПОЛУЧИТЬ ДОСТУП К PRIVATE-ПЕРЕМЕННОЙ КЛАССА? ЕСЛИ, ДА, ТО КАКИМ ОБРАЗОМ?**

Вообще доступ у приватной переменной класса можно получить только внутри класса, в котором она объявлена. Также доступ к приватным переменным можно осуществить через механизм Java Reflection API.

- **ДЛЯ ЧЕГО В ДЖАВЕ СТАТИЧЕСКИЕ БЛОКИ?**

Статические блоки в джава выполняются до выполнения конструктора, с помощью них инициализируют статические поля к примеру.

```
static final int i;
static{
    i=10;
}
```

Еще один ньюанс, блок статической инициализации может создаваться сам при компиляции программы:

Например:

```
public static int MAX = 100;
```

Будет создан код:

```
public static int MAX;
static{
    MAX = 100;
};
```

- **МОЖНО ЛИ ПЕРЕГРУЗИТЬ STATIC МЕТОД?**

Статические методы могут перегружаться нестатическими и наоборот – без ограничений. А вот в переопределении статического метода смысла нет.

- **РАССКАЖИТЕ ПРО ВНУТРЕННИЕ КЛАССЫ. КОГДА ВЫ ИХ БУДЕТЕ ИСПОЛЬЗОВАТЬ?**

Внутренний класс – это класс, который находится внутри класса или интерфейса. При этом он получает доступ ко всем полям и методам своего внешнего класса.
Для чего он может применятся? Например чтоб обеспечить какую-то дополнительную логику класса. Хотя использование внутренних классов усложняет программу, рекомендуется избегать их использование.

- **В ЧЕМ РАЗНИЦА МЕЖДУ ПЕРЕМЕННОЙ ЭКЗЕМПЛЯРА И СТАТИЧЕСКОЙ ПЕРЕМЕННОЙ? ПРИВЕДИТЕ ПРИМЕР.**

Статические переменные инициализируются при загрузке класса класслодером, и не зависят от объекта. Переменная экземпляра инициализируется при создании класса.
Пример: Например нам нужна глобальная переменная для всех объектов класс, например число посещений пользователей определенной статьи в интернете. При каждом новом посещении статьи создается новый объект и инкрементируется переменная посещений.

- **ПРИВЕДИТЕ ПРИМЕР КОГДА МОЖНО ИСПОЛЬЗОВАТЬ СТАТИЧЕСКИЙ МЕТОД?**

Статические методы могут быть использованы для инициализации статических переменных. Часто статические методы используются в классах утилитах, таких как Collections, Math, Arrrays

- **РАССКАЖИТЕ ПРО КЛАССЫ- ЗАГРУЗЧИКИ И ПРО ДИНАМИЧЕСКУЮ ЗАРУЗКУ КЛАССОВ.**

Любой класс, используемый в джава программу так или иначе был загружен в контекст программы каким-то загрузчиком.
Все виртуальные машины джава включают хотябы один загрузчик классов, так называем базовый загрузчик. Он загружает все основные классы, это классы из rt.jar. Интересно то, что этот загрузчик никак не связан с программой, тоесть мы не можем получить например у java.lang.Object имя зарузчика, метод getClassLoader() вернет нам null.

Следующий загрузчик – это загрузчик расширений, он загружает классы из $JAVA_HOME/lib/ext.

Далее по иерархии идет системный загрузчик, он загружает классы, путь к которым указан в переменно класпас.

Для примера предположим что у нас есть некий пользовательский класс MyClass и мы его используем. Как идет его загрузка… :

Сначала системный загрузчик пытается найти его в своем кэше загрузок его, если найден – класс успешно загружается, иначе управление загрузкой передается загрузчику расширений, он также проверяет свой кэш загрузок и в случае неудачи передает задачу базовому загрузчику. Тот проверяет кэш и в случае неудачи пытается его загрузить, если загрузка прошла успешно – загрузка закончена. Если нет – передает управление загрузчику расширений. Загрузчик расширений пытается загрузить класс и в случае неудачи передает это дело системному загрузчику. Системный загрузчик пытается загрузить класс и в случае неудачи возбуждается исключение java.lang.ClassNotFoundException.

Вот так работает загрузка классов в джава. Так называемое делегирование загрузки.

Если в системе присутствуют пользовательские загрузики, то они должны быть унаследованы от класса java.lang.ClassLoader .

Что же такое статическая и что такое динамическая загрузка класса?

Статическая загрузка класса происходит при использовании оператора "new".

Динамическая загрузка происходит "на лету" в ходе выполнения программы с помощью статического метода класса Class.forName(имя класса). Для чего нужна динамическая загрузка? Например мы не знаем какой класс нам понадобится и принимаем решение в ходе выполнения программы передавая имя класса в статический метод forName().

- **ДЛЯ ЧЕГО НУЖЕН ОПЕРАТОР "ASSERT" В ДЖАВА?**

Это так называемый оператор утверждений. Он проверяет некое условие, если оно ложно, то генерируется AssertationError

```
assert status: "message error";
```

Тут проверяется булевская переменная "status".

- **ПОЧЕМУ В НЕКОТОРЫХ ИНТЕРФЕЙСАХ ВООБЩЕ НЕ ОПРЕДЕЛЯЮТ МЕТОДОВ?**

Это так называемые интерфейсы – маркеры. Они просто указывают что класс относится к определенной группе классов. Например интерфейс Clonable указывает на то, что класс поддерживает механизм клонирования.

Степень абстракции в данном случае доведен до абсолюта. В интерфейсе вообще нет никаких объявлений.

Интерфейси-маркери в Java:

- Searilizable interface
- Cloneable interface
- Remote interface

- ThreadSafe interface

- **КАКАЯ ОСНОВНАЯ РАЗНИЦА МЕЖДУ STRING, STRINGBUFFER, STRINGBUILDER?**

String – неизменяемый класс, тоесть для для добавление данных в уже существующую строку, создается новый объект строки.
StringBuffer и StringBuilder могут изменятся и добавление строки не такое дорогостоющее с точки зрения памяти. Первы – синхронизированный, второй – нет. Это их единственное различие.
Правда если нам нужно сделать подстроку строки, то лучше использовать String, так как ее массив символов не меняется и не создается заново для новой строки. А вот в StringBuffer и StringBuilder для создания подстроки создается новый массив символов.

- **РАССКАЖИТЕ ПРО ПОТОКИ ВВОДА-ВЫВОДА JAVA.**

Потоки ввода-вывода бывают двух видов:
- байтовый поток(InputStream и OutputStream);
- символный поток(Reader и Writer);

Это все абстрактные классы – декораторы, которым можно добавлять дополнительный функционал, например:
InputStream in = new FileInputStream(new File("file.txt"));

- **ЧТО ТАКОЕ HEAP И STACK ПАМЯТЬ В JAVA?**

Java Heap (куча) – динамически распредляемая область памяти, создаваемая при старте JVM. Используется Java Runtime для выделения памяти под объекты и JRE классы. Создание нового объекта также происходит в куче. Здесь работает сборщик мусора: освобождает память путем удаления объектов, на которые нет каких-либо ссылок. Любой объект, созданный в куче, имеет глобальный доступ и на него могут ссылаться с любой части приложения.
Строгими тезами:
- Все обьекты обитают в куче и попадают туда при создании.
- обьект состоит из полей класса и методов.
- в куче выделяется место под сам обьект, количество выделенной памяти зависит от полей, если у тебя полем класса, к примеру, служит интовая переменная, то не важно, инициализируешь ты ее как "0" или как "1000000" – обьект займет в куче свои биты, + столько байт сколько вмещает тип int(+32 бита), и так с каждым полем.

Стековая память в Java работает по схеме LIFO (Последний-зашел-Первый-вышел). Всякий раз, когда вызывается метод, в памяти стека создается новый блок, который содержит примитивы и ссылки на другие объекты в методе расположение в RAM и достижение процессору через указатель стека. Как только метод заканчивает работу, блок также перестает использоваться, тем самым предоставляя доступ для следующего метода. Размер стековой памяти намного меньше объема памяти в куче.

Строгими тезами:

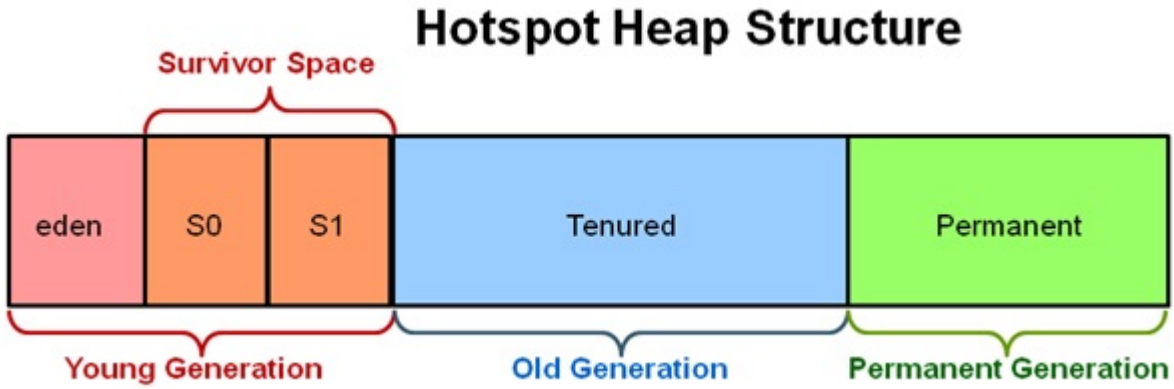
- Все методы обитают в стеке и попадают туда при вызове.
- Переменные в методах так же имеют стековую память, по скольку они локальные.
- Если в методе создается обьект, то он помещается в кучу, но его ссылка все еще будет находится в стеке и после того как метод покинет стек - обьект станет жертвой сборщика мусора, так как ссылка на него утеряна, и из главного стека программы невозможно будет добраться до такого обьекта.

- **КАКАЯ РАЗНИЦА МЕЖДУ STACK И HEAP ПАМЯТЬЮ В JAVA?**

Приведем следующие различия между Heap и Stack памятью в Java.

- Куча используется всеми частями приложения в то время как стек используется только одним потоком исполнения программы.
- Всякий раз, когда создается объект, он всегда хранится в куче, а в памяти стека содержится ссылка на него. Память стека содержит только локальные переменные примитивных типов и ссылки на объекты в куче.
- Объекты в куче доступны с любой точки программы, в то время как стековая память не может быть доступна для других потоков.
- Управление памятью в стеке осуществляется по схеме LIFO.
- Стековая память существует лишь какое-то время работы программы, а память в куче живет с самого начала до конца работы программы.
- Мы можем использовать -Xms и -Xmx опции JVM, чтобы определить начальный и максимальный размер памяти в куче. Для стека определить размер памяти можно с помощью опции -Xss .
- Если память стека полностью занята, то Java Runtime бросает java.lang.StackOverflowError, а если память кучи заполнена, то бросается исключение java.lang.OutOfMemoryError: Java Heap Space.
- Размер памяти стека намного меньше памяти в куче. Из-за простоты распределения памяти (LIFO), стековая память работает намного быстрее кучи.

- **РАССКАЖИТЕ ПРО МОДЕЛЬ ПАМЯТИ В ДЖАВА?**



В Джаве память устроена следующим образом, есть два вида:

- куча
- стек

Куча состоит из статического контекста и самой кучи

Перейдем к куче. Куча состоит из двух частей:

- Новая куча

- Старая куча

Новая куча в свою очередь состоит из двух частей:

- Eden(назовем ее первая) куча
- Survival(выжившая) куча

Краткое описание:

- Eden Space (heap) – в этой области выделятся память под все создаваемые из программы объекты. Большая часть объектов живет недолго (итераторы, временные объекты, используемые внутри методов и т.п.), и удаляются при выполнении сборок мусора это области памяти, не перемещаются в другие области памяти. Когда данная область заполняется (т.е. количество выделенной памяти в этой области превышает некоторый заданный процент), GC выполняет быструю (minor collection) сборку мусора. По сравнению с полной сборкой мусора она занимает мало времени, и затрагивает только эту область памяти – очищает от устаревших объектов Eden Space и перемещает выжившие объекты в следующую область.
- Survivor Space (heap) – сюда перемещаются объекты из предыдущей, после того, как они пережили хотя бы одну сборку мусора. Время от времени долгоживущие объекты из этой области перемещаются в Tenured Space.
- Tenured (Old) Generation (heap) – Здесь скапливаются долгоживущие объекты (крупные высокоуровневые объекты, синглтоны, менеджеры ресурсов и проч.). Когда заполняется эта область, выполняется полная сборка мусора (full, major collection), которая обрабатывает все созданные JVM объекты.
- Permanent Generation (non-heap) – Здесь хранится метаинформация, используемая JVM (используемые классы, методы и т.п.).

- **КАК РАБОТАЕТ СБОРЩИК МУСОРА (GARBAGE COLLECTOR)?**

Во-первых что стоит сказать, что у сборщика мусора есть несколько алгоритмов работы, он не один.

Когда происходит очистка памяти? Если память в Первой куче полностью заполнена, то туда идет сборщик мусора и делает свою работу) Какую именно, зависит от обстоятельств… Например если в первой кучи много мусора(т.е. объектов с нулевой ссылкой), то сборщик мусора помечает эти объекты, далее те что остались объекты со ссылками он их переносит в Выжившую кучу, а в первой куче он просто все удаляет.

Ситуация другая, в первой кучи мало мусора, но очень много рабочих объектов. Как поступает в этом случае сборщик мусора?

Он помечает мусор, удаляет его и оставшиеся объекты компонует.

Также следует заметить что при нехватке места в Выжившей куче, объекты переносятся в старую кучу, там хранятся как правило долго живущие объекты.

Также следует заметить что сборщик мусора вызывается сам периодически, а не только когда памяти не хватает.

- **РАССКАЖИТЕ ПРО ПРИВЕДЕНИЕ ТИПОВ. ЧТО ТАКОЕ ПОНИЖЕНИЕ И ПОВЫШЕНИЕ ТИПА? КОГДА ВЫ ПОЛУЧАЕТЕ CLASSCASTEXCEPTION?**

Приведение типов это установка типа переменной или объекта отличного от текущего. В ждава есть два вида приведения:

- автоматическое

- не автоматическое

Автоматическое происходит например:
byte-> short->int->long->float->double
тоесть если мы расширяем тип, то явное преобразование не требуется, приведение происходит автоматически. Если же мы сужаем, то необходимо явно указывать приведение типа.
В случае же с объектами, то мы можем сделать автоматическое приведение от наследника к родителю, но никак не наоборот, тогда вылетит ClassCastException.

- **ЧТО ТАКОЕ СТАТИЧЕСКИЙ КЛАСС, КАКИЕ ОСОБЕННОСТИ ЕГО ИСПОЛЬЗОВАНИЯ?**

Статическим классом может быть только внутренний клас(определение класса размещается внутри другого класса). В объекте обычного внутреннего класса хранится ссылка на объект внешнего класса. Внутри статического внутреннего класса такой ссылки нет.
То есть: Для создания объекта статического внутреннего класса не нужен объект внешнего класса. Из объекта статического вложенного класса нельзя обращаться к нестатическим членам внешнего класса напрямую. И еще обычные внутренние классы не могут содержать статические методы и члены.
Зачем вообще нужны внутренние классы? – Каждый внутренний класс способен независимо наследовать определенную реализацию. Таким образом внутренний класс не ограничен при наследовании в ситуациях, когда внешний класс уже наследует реализацию. То есть это как бы вариант решения проблемы множественного наследования.

- **КАКИМ ОБРАЗОМ ИЗ ВЛОЖЕННОГО КЛАССА ПОЛУЧИТЬ ДОСТУП К ПОЛЮ ВНЕШНЕГО КЛАССА.**

Если класс внутренний то: Внешнийкласс.this.Поле внешнего класса Если класс статический внутренний(вложенный),то в методе нужно создать объект внешнего класса, и получить доступ к его полю.Или второй вариант объявить это поле внешнего класса как static

- **КАКИЕ СУЩЕСТВУЮТ ТИПЫ ВЛОЖЕННЫХ КЛАССОВ? ДЛЯ ЧЕГО ОНИ ИСПОЛЬЗУЮТСЯ?**

Вложенные классы существуют внутри других классов. Нормальный класс – полноценный член пакета. Вложенные классы, которые стали доступны начиная с Java 1.1, могут быть четырех типов:
- статические члены класса
- члены класса
- локальные классы

- анонимные классы

Статические члены классов (static nested classes) – как и любой другой статический метод, имеет доступ к любым статическим методам своего внешнего класса, в том числе и к приватным. К нестатическим полям и методам обрамляющего класса он не может обращатся напрямую. Он может использовать их только через ссылку на экземпляр класса родителя.

Члены класса – локальные классы, объявленные внутри блока кода. Эти классы видны только внутри блока.

Анонимные классы – Эти типы классов не имеют имени и видны только внутри блока.