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

vasyl1889

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).

 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.

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

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.

Cтрогими 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.

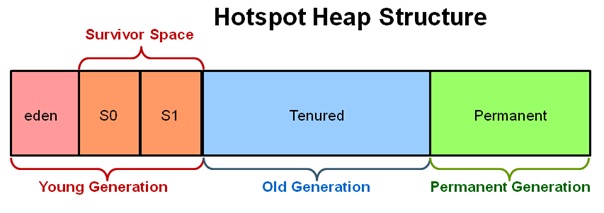
Стековая память в Java работает по схеме LIFO (Последний-зашел-Первый-вышел). Всякий раз, когда вызывается метод, в памяти стека создается новый блок, который содержит примитивы и ссылки на другие объекты в методе расположение в RAM и достижение процессору через указатель стека. Как только метод заканчивает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

Cтрогими 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?**

[](https://3.bp.blogspot.com/-btZICcb7S_c/V0yOYppvPOI/AAAAAAAAAx8/QBEUPQSC4jEmKlAEVvEhn-3dugnC_SxlACLcB/s1600/memory%2Bmodel.jpg)

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.