**Готовимся к собеседованию**

**Основы**

**Что такое JVM, JDK, JRE, в чем отличия?**

**jvm - Java Virtual Machine - среда ("виртуальная"), в которой выполняется твой скомпилированный код**

**jre - Java Runtime Environment. Просто пакет, который включает в себя JVM и минимальный набор библиотек для работы программ. И еще браузерный плагин, где будут выполняться апплеты.**

**jdk - Java Development Kit - это уже полноценный набор библиотек и инструментов для для создания, компилирования и дебага программ. Включает в себя jre**

* **Какие типы данных поддерживает Java? Что такое автоупаковка и автораспаковка? // wrapper**

**Примитивные**

* byte (целые числа, 1 байт, [-128, 127])
* short (целые числа, 2 байта, [-32768, 32767])
* int (целые числа, 4 байта, [-2147483648, 2147483647])
* long (целые числа, 8 байт, [-922372036854775808,922372036854775807])
* float (вещественные числа, 4 байта)
* double (вещественные числа, 8 байт)
* char (символ Unicode, 2 байта, [0, 65536])
* boolean (значение истина/ложь, используется int, зависит от JVM)

**и Ссылочные – все объекты которые мы создаем.**

**Автоупаковка и распаковка - это функция преобразования примитивных типов в объектные и наоборот.**

**Автоупаковка**

Integer integer = 9;

**Распаковка**

int in = 0;

in = new Integer(9);

**Автоупаковка применяется компилятором Java в следующих условиях:**

**Когда значение примитивного типа передается в метод в качестве параметра метода, который ожидает объект соответствующего класса-оболочки.**

**Когда значение примитивного типа присваивается переменной, соответствующего класса оболочки.**

**Распаковка применяется компилятором Java в следующих условиях:**

**Когда объект передается в качестве параметра методу, который ожидает соответствующий примитивный тип.**

**Когда объект присваивается переменной соответствующего примитивного типа.**

* **Что такое String[] args в методе main()?**

**Это строковые аргументы, которые подаются на вход программы через консоль**

* **Какими значениями по умолчанию инициализируются локальные переменные, и поля объектов?**

**Локальные переменные – это переменные внутри методов, и они не инициализируются изначально. Поля объектов инициализируются значениями по умолчанию – для числовых типов – 0, для ссылочных типов – null, для char – символ с кодом 0, для Boolean – false.**

* **Почему Java не поддерживает указатели, множественное наследование?**

**С целью безопасности кода и снятия части проблем. Множественное наследование не используется с целью однозначности работы наследуемых классов.**

* **Что такое циклы, какие типы циклов вы знаете?**

В Java используются циклы for, while, do-while.

**while(условие) {**

**// тело цикла**

**}**

**do {**

**// тело цикла**

**}**

**while(условие-логическое выражение)**

**for(инициализация; логическое выражение (условие); шаг (итерация)) {**

**// тело цикла**

**}**

**for(Object object : objects) {**

**// тело цикла**

**}**

* **Как можно организовать бесконечный цикл?**

**While(true) и for(;;)**

* **Что такое break, continue и label?**

**Break – прерывание цикла и выход из него;**

**Continue – продолжение цикла с пропуском кода расположенного ниже;**

**Label – метка, которая указывается перед циклом для быстрого перехода с помощью оператора break labelName.**

* **Что такое пакеты(packages), зачем они нужны, какие пакеты по умолчанию подключены?**

**Логическое разделение файлов исходного проекта, его структура, а также разделение классов, которые могу иметь одинаковые имена. Подключен по умолчанию java.util.lang (Например String)**

* **При импорте пакета, импортируются ли классы из подпакетов?**

**Нет**

* **Что такое статический импорт?**

**Импорт статических методов или элементов перечислений.**

**Конструкция статического импорта позволяет получить прямой доступ к статическим членам без необходимости наследования от того типа, который содержит эти статические члены. Вместо этого, программа импортирует члены индивидуально:**

**import static java.lang.Math.PI;**

**или все целиком:**

**import static java.lang.Math.\*;**

**Однажды импортированный статический член может быть использован без указания имени класса:**

**double r = cos(PI \* theta);**

**Объявление статического импорта аналогично объявлению обычного импорта.**

**ООП**

* **Что такое классы/объекты?**

**Класс – это описатель общих свойств группы объектов. Этими свойствами могут быть как характеристики объектов (размер, вес, цвет и т.п.), так и поведения, роли и т.п.**

**Объекты – это экземпляры классов.**

* **Что такое инкапсуляция, наследование и полиморфизм?**

**Инкапсуляция** — свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. Для Java корректно будет говорить, что инкапсуляция — это «сокрытие реализации». Пример из жизни — пульт от телевизора. Мы нажимаем кнопочку «увеличить громкость» и она увеличивается, но в этот момент происходят десятки процессов, которые скрыты от нас. Для Java: можно создать класс с 10 методами, например, вычисляющие площадь сложной фигуры, но сделать из них 9 private. 10й метод будет называться «вычислитьПлощадь()» и объявлен public, а в нем уже будут вызываться необходимые скрытые от пользователя методы. Именно его и будет вызывать пользователь.

**Наследование** — свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс — потомком, наследником, дочерним или производным классом.

**Полиморфизм**— свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Иными словами - это способность (в программировании) представить тот же интерфейс для различных форм (типов данных), лежащих в основе. Это означает, что классы имеют различную функциональность, несмотря на то, что используют общий интерфейс и могут быть вызваны динамически через специальную ссылку класса.

Ну и совсем просто - Способность к изменению функциональности, унаследованной от базового класса, называется **полиморфизмом**

* **Что такое абстрактный класс?**

**Абстракция** —  означает выделение значимой информации и исключение из рассмотрения незначимой. С точки зрения программирования — это правильное разделение программы на объекты. Абстракция позволяет отобрать главные характеристики и опустить второстепенные.

Пример: описание должностей в компании. Здесь название должности значимая информация, а описание обязанностей у каждой должности — это второстепенная информация. К примеру главной характеристикой для «директора» будет то, что это должность чем-то управляет, а чем именно (директор по персоналу, финансовый директор, исполнительный директор) это уже второстепенная информация.

**Абстрактный класс, как раз тот класс, который реализует абстракцию. Важное замечание – вы не можете создавать экземпляры абстрактного класса, а только расширять.**

* **Можно ли 1. Создавать абстрактные методы в обычном классе, 2. Создавать обычное методы в абстрактном классе**

1. **Нет**
2. **Да**

* **Что такое интерфейс?**

**Ключевое слово interface используется для создания полностью абстрактных классов. Создатель интерфейса определяет имена методов, списки аргументов и типы возвращаемых значений, но не тела методов.**

**Наличие слова interface означает, что именно так должны выглядеть все классы, которые реализуют данный интерфейс. Таким образом, любой код, использующий конкретный интерфейс, знает только то, какие методы вызываются для этого интерфейса, но не более того.**

**public interface** SomeName{  
 **void** method();  
 **int** getSum();  
}

* **Бывают ли пустые интерфейсы? Если да, то зачем они нужны?**

**Да. Называются маркерными (как например Serializable)**

* **Можно ли создать объект Flyable f = ...;? Flyable – интерфейс**

**Нет, мы не можем создавать объекты из интерфейсов, но можем создавать объекты класса, реализующего указанный интерфейс**

* **Что означает ключевое слово static? Можно ли переопределить (override) private или static методы?**

**Static поля – это поле класса, которое привязано именно к классу, а не к его экземпляру**

**Static метода – для работы метода нам не нужен объект.**

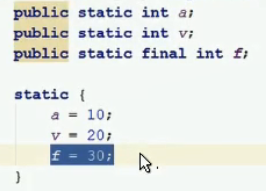
**Мы не можем переопределить ни private, ни static методы. Static не относится к объекту, а Private не виден из объекта наследника.**

* **Можно ли получить доступ к не статическому полю из статического метода?**

**Нет, по той причине, что не статическое поле доступно только в объекте.**

* **Что такое блок static { }?**

**Блок, в котором выполняется код при первом обращении к классу.**



* **Что такое переопределение и перегрузка методов?**

**Java позволяет создавать несколько методов с одинаковыми именами, но разными сигнатурами. Создание метода с тем же именем, но с другим набором параметров называется перегрузкой. Какой из перегруженных методов должен выполняться при вызове, Java определяет на основе фактических параметров.**

**public void** method() { }  
**public void** method(**int** a) { }  
**public int** method(String str) { **return** 0;}  
**public float** method(Double str) { **return** 0;}

**Метод в классе-наследнике, совпадающий по сигнатуре с методом из родительского класса называется переопределенным методом. Переопределить базовый статический метод нельзя: Instance method имяМетода in классНаследник cannot override method имяМетода in родительскийКласс**

* **Что такое конструктор и перегрузка конструктора? Может ли класс иметь несколько конструкторов?**

**Конструктор — это специальный метод, который вызывается при создании нового объекта. Конструктор инициализирует объект непосредственно во время создания. Имя конструктора совпадает с именем класса, включая регистр, а по синтаксису конструктор похож на метод без возвращаемого значения.**

**public class** Circle {  
  
 **public** Circle() {  
 }  
}

**Перегрузка конструктора аналогична перегрузки методов. Класс может иметь несколько конструкторов.**

* **Когда вызывается конструктор?**

**Конструктор вызывается тогда, когда создается объект.**

* **Какие модификаторы доступа вы знаете и сколько их?**

**private, default, protected, public**

* **Что означает ключевое слово this?**

**Это ссылка объекта на самого себя внутри объекта**

* **Что означает ключевое слово super?**

**Это ссылка на родительский класс**

* **Что означает ключевое слово final?**

**Это обозначение константы для поля, для метода, то что его нельзя переопределить, для класса – то что он становится недоступным для наследования.**

* **Какой класс является базовым для всех классов?**

**Object**

* **В чем разница между String, StringBuffer, StringBuilder?**

**String является неизменяемой, при «изменении» создается новый объект в памяти с переприсвоением ссылки.**

**StringBuffer – мутабельный класс для использования в многопоточном коде.**

**StringBuilder - мутабельный класс для однопоточного выполнения.**

* **Может ли метод быть одновременно abstract и final?**

**Нет. Получите: Illegal combination of modifiers: ‘abstract’ and ‘static’. Модификатор abstract говорит, что метод будет реализован в другом классе, а static наоборот указывает, что этот метод будет доступен по имени класса.**

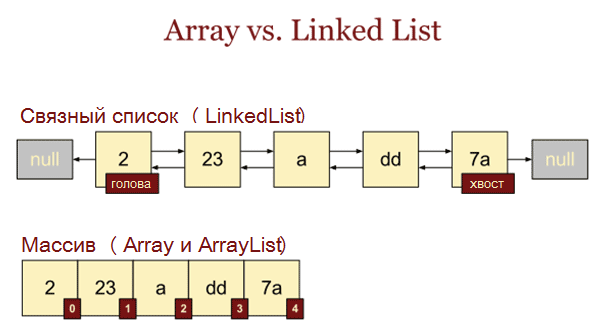
* **Что такое cast?**

**Это уточнение типа получаемого результата. Например, мы можем в Object присвоить любой объект, но, чтобы добраться до его методов и полей, нам нужно будет «уточнить» тип данных или выполнить cast.**

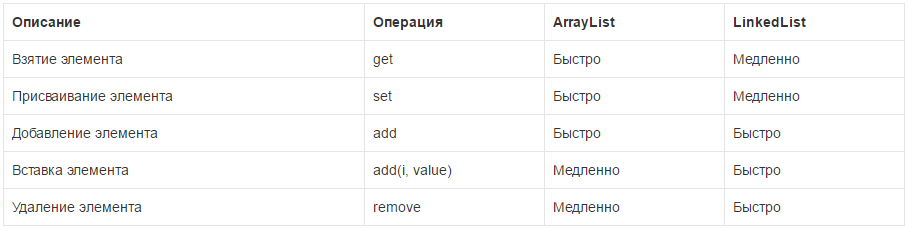
**Коллекции**

* **Что такое ArrayList и LinkedList, какая между ними разница?**

**ArrayList реализован внутри в виде обычного массива. Поэтому при вставке элемента в середину, приходится сначала сдвигать на один все элементы после него, а уже затем в освободившееся место вставлять новый элемент. Зато в нем быстро реализованы взятие и изменение элемента – операции get, set, так как в них мы просто обращаемся к соответствующему элементу массива. Работает только со ссылочными типами данных.**



**LinkedList реализован внутри по-другому. Он реализован в виде связного списка: набора отдельных элементов, каждый из которых хранит ссылку на следующий и предыдущий элементы. Чтобы вставить элемент в середину такого списка, достаточно поменять ссылки его будущих соседей. А вот чтобы получить элемент с номером 130, нужно пройтись последовательно по всем объектам от 0 до 130.**



**Для Большого количества выполнения операций добавления и удаления следует использовать LinkedList.**

* **Чем массив отличается от ArrayList?**

**Массив является статическим, он никак не меняет свой размер, в отличии от ArrayList**

* **В чем разница между ArrayList и Vector?**

**Vector – синхронизированная версия ArrayList, его используют для работы в многопоточности.**

* **Что такое Set, HashSet, LinkedHashSet, TreeSet, и в чем между ними разница?**

**Set – интерфейс для всех Сетов**

**HashSet – хранит в себе только уникальные значения, НЕ гарантирует порядок хранения**

**LinkedHashSet – Хранит в себе только уникальные значения, ГАРАНТИРУЕТ порядок хранения, запоминает порядок добавления элементов.**

**TreeSet – отсортированное множество. Для сортировки значений использует интерфейс Comparable.**

* **В чем разница между Hashtable, HashMap, ConcurrentHashMap, Collections.synchronizedMap(…)?**

**HashMap – стандартная коллекция ключ-значение, плохо применима для работы в многопоточности при наличии операций записи.**

**Hashtable – синхронизированный вариант HashMap, но с синхронизацией по типу доступа на запись. Т.е. при многопоточном обращении к этой коллекции мы получим очередь из потоков, которые собираются что-то записать, т.к. одновременно может писать только один поток. Hashtable не позволяет иметь null ключи или значения. HashMap позволяет иметь один null ключ и сколько угодно null значений.**

**ConcurrentHashMap – самый оптимальный вариант для работы с многопоточностью, т.к. тут свободное чтение, а при записи блокируется только часть таблицы.**

**Collections.synchronizedMap – самый худший вариант, т.к. это класс-обертка над коллекцией, которая дает возможность использования его в многопоточности, но это достается ценой синхронизации по объекту, т.е. одновременно читать или писать в мэп обернутый в данную обертку может только один поток.**

* **Существуют ли какие-нибудь требованию к объектам, используемым в качестве ключа в HashMap?**

**Если объекты вашей программы будут использованы в виде ключа к HashMap, то вам нужно будет переписать метод equals и hashCode у вашего класса. При этом equals и hashCode должны быть согласованы между собой, т.е. при equals = true hashCode должен быть одинаковым!**

* **Что такое hashCode()? Что будет если его не переопределить?**

**Это метод, который преобразует наш объект в число типа int по какому-нибудь алгоритму. Обычно этот метод переопределяется в самом классе, если его не переопределить, то по умолчанию он будет работать так, как нам скажет Object. Object вычисляет hashCode на основании нахождения объекта в какой-либо ячейки памяти. Т.е. при одинаковых по equals объектах hashCode будет разный.**

* **Что такое loadFactor и initialCapactity в HashMap?**

**В HashMap есть loadFactor(фактор загрузки, при котором происходит увеличение емкости коллекции (количества корзин) в два раза и произошел пересчет хеш значений для нашей коллекции, значение по умолчанию – 0,75) и initialCapactity(начальная емкость, она должна быть 2^N, по умолчанию - 16)**

* **Что такое коллизии и как их можно избежать?**

**Когда несколько записей пытаются попасть в одну ячейку. Мы должны пытаться их избежать, потому что если у нас в одну ячейку попадет много записей, то выстроится длинный linkedList и мы будем терять в скорости поиска по hashMap’у, его основном преимуществе. Избежать коллизии можно меняя алгоритм, по которому рассчитывается hashCode.**

* **Что такое Iterator, чем он отличается от ListIterator?**

**Iterator – класс, который позволяет обходить коллекции и модифицировать на ходу, удалять элементы.**

**ListIterator – итератор, с которым мы можем работать в List’ах. Разница в том, что он может добавлять записи в лист, а также двигаться в двух направлениях, как вперед, так и назад.**

**Iterator может использоваться для перебора элементов Set, List и Map. В отличие от него, ListIterator может быть использован только для перебора элементов коллекции List**

**Iterator позволяет перебирать элементы только в одном направлении, при помощи метода next(). Тогда как ListIterator позволяет перебирать список в обоих направлениях, при помощи методов next() и previous()**

**При помощи ListIterator вы можете модифицировать список, добавляя/удаляя элементы с помощью методов add() и remove(). Iterator не поддерживает данного функционала**

**Исключения**

* **Что означает каждое из ключевых слов: try, catch, finally, throw, throws?**

**Try – попытаться выполнить код**

**Catch – перехватить исключение, при его возникновении**

**Finally – гарантированно выполняемый код после try, вне зависимости от возникновении исключительной ситуации.**

**Throw – бросить новое исключение.**

**Throws – пробросить исключение наверх из метода. Т.е. возникающие в коде исключения мы не обрабатываем потому как не имеем возможности и пробрасываем его наверх. Записывается как:**

**тип имяМетода(список аргументов) throws список исключений {}**

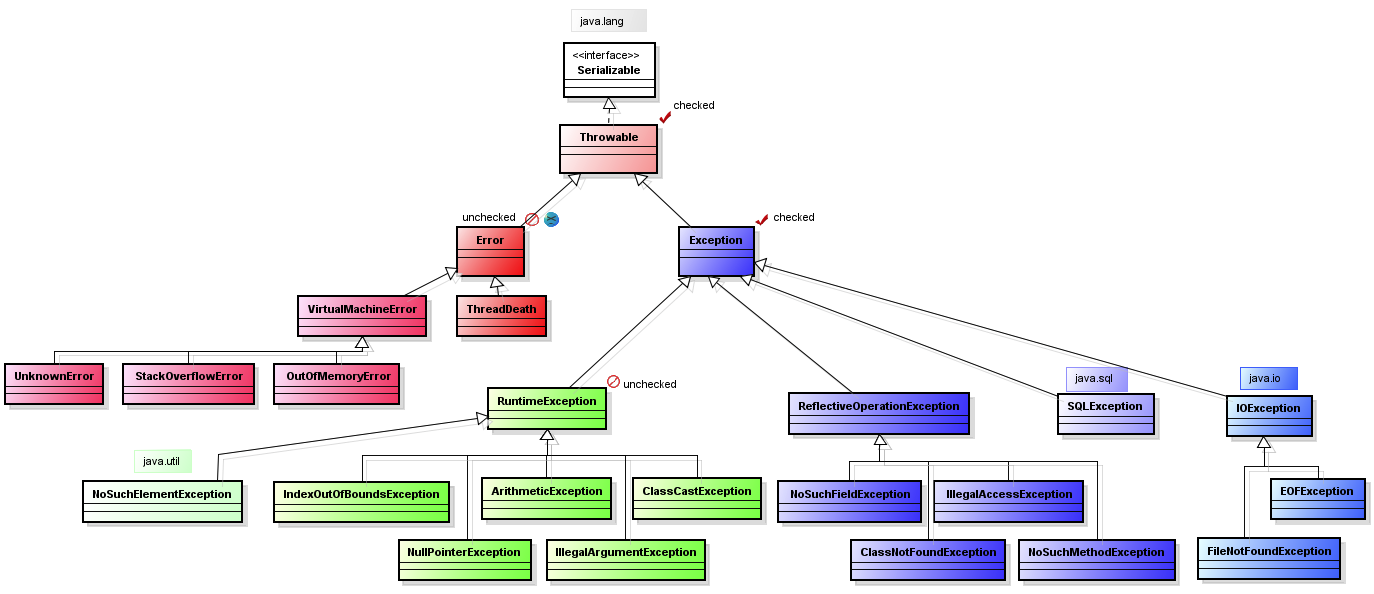
* **Для чего в основном используется блок finally?**

**Основное использование – освобождение ресурсов, закрытие файлов, соединений.**

* **Что такое try-with-resource?**

**Это краткая запись try finally, которая не требует ручного закрытия/освобождения ресурсов. В начале блока ресурсы открываются, при окончании работы кода в блоке, они (ресурсы) автоматически освобождаются.**

* **Покажите примерную схему дерева классов, наследующихся от Throwable?**



* **Что такое checked и unchecked исключения?**

**Checked – исключения, которые мы обязаны перехватить в нашем коде**

**Unchecked – исключения, которые мы не обязаны перехватить, но они могут возникнуть.**

* **Что происходит при необработанном исключении (мы его не перехватили)? \*насчет потоков\***

**Если мы не перехватываем исключение, которое возникло оно долетает до стандартного обработчика и тогда программа завершает свою работу. Если вы создаёте потоки, то из потоков исключения не вылетают, чтобы перехватить исключения из потоков, нам нужен ExecutorService, который позволяет узнать, было ли исключение в каком-либо из потоков или нет. Исключение, которое возникло в потоке останавливает только поток, не влияя на родительский поток.**

* **В чем разница между Exception и Error?**

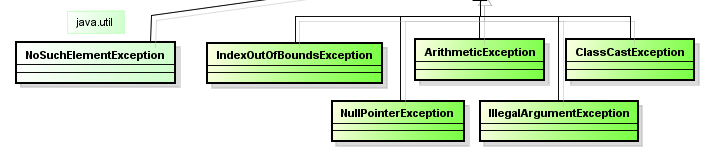
**Exception – мы можем перехватить и обработать, с Error мы не можем бороться, т.е. это гарантированное завершение работы.**

* **Может ли исключение быть снова брошено?**

**Может, при обработке исключения мы можем его перехватить и бросить снова или вызвать вместо полученного исключения новое исключение.**

* **Приведите примеры подклассов RuntimeException?**

**Это все идет из иерархии Throwable**



* **В чем смысл проброса исключений?**

**Может возникнуть ситуация, когда мы не можем в конкретной части кода, в данном методе корректно обработать исключение, просто нет возможности, поэтому нам нужно пробросить исключения выше.**

**Многопоточность**

* **В чем разница между процессом и потоком?**

**Процесс – это, можно считать, что это отдельная программа. Если более подробно:**

**Процесс — это совокупность кода и данных, разделяющих общее виртуальное адресное пространство. Процессы изолированы друг от друга, поэтому прямой доступ к памяти чужого процесса невозможен (взаимодействие между процессами осуществляется с помощью специальных средств). Для каждого процесса ОС создает так называемое «виртуальное адресное пространство», к которому процесс имеет прямой доступ. Это пространство принадлежит процессу, содержит только его данные и находится в полном его распоряжении. Операционная система же отвечает за то, как виртуальное пространство процесса проецируется на физическую память.**

**Поток – потоки внутри самой программы. Подробнее:**

**Один поток («нить» или «трэд») – это одна единица исполнения кода. Каждый поток последовательно выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса.**

* **Какие существуют способы создания потоков?**

**Существует несколько способов создания.**

**- С помощью класса, реализующего Runnable**

**- Создать объект класса Thread.**

**- Создать объект класса, реализующего интерфейс Runnable**

**- С помощью класса, расширяющего Thread**

**Создать объект класса ClassName extends Thread.**

**Переопределить run() в этом классе**

**- С помощью класса, реализующего java.util.concurrent.Callable**

**- Создать объект класса, реализующего интерфейс Callable**

**- Создать объект ExecutorService с указанием пула потоков.**

**- Создать объект Future.**

* **Что такое Deadlock?**

**Deadlock, он же взаимная блокировка, явление при котором все потоки находятся в режиме ожидания. Чтобы уменьшить шанс появления deadlock’a не рекомендуется использовать методы wait() и notify().**

* **В чем разница между notify() и notifyAll()?**

**Метод notify пробуждает один из потоков, который вызвал метод wait() у этого монитора. Метод notifyAll пробуждает все потоки. Очередность выполнения в этом случае будет определяться приоритетом потока.**

* **В чем разница между sleep() и wait()?**

**Sleep – просто усыпляет поток на определенное время.**

**Wait – переводит поток в режим ожидания пока по этому монитору не скажут notify() или notifyAll().**

* **Можно ли перезапустить завершившийся поток?**

**Перезапустить завершившийся поток нельзя! Только создать новый.**

* **Какие способы синхронизации вы знаете, что может выступать в роли монитора?**

**Способы синхронизации:**

**1. Системная синхронизация с использованием wait/notify.**

**Поток, который ждет выполнения каких-либо условий, вызывает у этого объекта метод wait, предварительно захватив его монитор. На этом его работа приостанавливается. Другой поток может вызвать на этом же самом объекте метод notify (опять же, предварительно захватив монитор объекта), в результате чего, ждущий на объекте поток "просыпается" и продолжает свое выполнение.**

**2. Системная синхронизация с использованием join**

**Метод join, вызванный у экземпляра класса Thread, позволяет текущему потоку остановиться до того момента, как поток, связаный с этим экземпляром, закончит работу.**

**3. Использование классов из пакета java.util.concurrent,**

**который предоставляет набор классов для организации межпоточного взаимодействия. Примеры таких классов - Lock, семафор (Semaphore), etc. Концепция данного подхода заключается в использовании атомарных операций и переменных.**

**В роли монитора может выступать:**

**- самый обычной объект(Object), когда мы делаем синхронизацию по объекту.**

**- объект, у которого вызвали синхронизированный метод**

**- класс, у которого вызвали статический метод**

* **Зачем нужен Semaphore, CyclicBarrier, CountDownLatch, ExecutorService?**

**Semaphore – Служит для ограничения доступа определенного количества потоков к критическому блоку. Т.е. указывает, какое максимальное количество потоков может исполнять отмеченный блок одновременно.**

**CyclicBarrier – Позволяет тормозить все потоки, пока заданное количество потоков не подойдут к определенной черте или завершат свою работу с вызовом метода await().**

**CountDownLatch – (замок с обратным отсчетом) предоставляет возможность любому количеству потоков в блоке кода ожидать до тех пор, пока не завершится определенное количество операций, выполняющихся в других потоках, перед тем как они будут «отпущены», чтобы продолжить свою деятельность. В конструктор CountDownLatch (CountDownLatch(int count)) обязательно передается количество операций, которое должно быть выполнено, чтобы замок «отпустил» заблокированные потоки. Блокировка потоков снимается с помощью счётчика: любой действующий поток, при выполнении определенной операции уменьшает значение счётчика. Когда счётчик достигает 0, все ожидающие потоки разблокируются и продолжают выполняться (примером CountDownLatch из жизни может служить сбор экскурсионной группы: пока не наберется определенное количество человек, экскурсия не начнется).**

**ExecutorService – Занимается быстрым запуском потоков, т.е. он подготавливает потоки и может быстро раздать им задачи. Служит для повышения производительности работы с большим количеством потоков.**

* **Что такое daemon thread?**

**Потоками-демонами называются потоки, работающие в фоновом режиме для нашей программы.**

**В Java процесс завершается тогда, когда завершается последний его поток. Даже если метод main() уже завершился, но еще выполняются порожденные им потоки, система будет ждать их завершения.**

**Однако это правило не относится к особому виду потоков – демонам. Если завершился последний обычный поток процесса, и остались только потоки-демоны, то они будут принудительно завершены и выполнение процесса закончится. Чаще всего потоки-демоны используются для выполнения фоновых задач, обслуживающих процесс в течение его жизни.**

* **Какие преимущества дает использование Lock по сравнению с synchronized blocks?**

**Существует несколько видов замков – ReentrantLock и ReentarntReadWriteLock. Указанные замки мы можем захватывать в одном методе, а в другом методе мы можем их освободить, т.е. замок не привязан к конкретному блоку кода. Кроме того, мы можем проверить замок на захват, т.е. когда поток подойдет к критической секции, то он гарантированно перейдет в режим ожидания момента, пока секция не будет свободна. Замок, также, можно попытаться захватить.**

**Преимуществом ReentarntReadWriteLock является разделение потоков на чтение и запись. Пока выполняется чтение ресурса, заблокированного на чтение, ни один другой поток писать не может и в обратной ситуации – когда поток приступил к операции записи и установил замок записи, ни один поток не может читать. Таким образом осуществляется непротиворечивость данных. Так же, одновременно читать может неограниченное число потоков, писать может только один поток.**

**JDBC**

* **Что такое JDBC?**

**Мост между нашим приложением и базой данных**

* **Какова роль драйвера в JDBC?**

**Драйвер реализует набор интерфейсов для взаимодействия с БД. В нем прописано, как должна наша программа взаимодействовать с какой-либо базой данных (SQLite, MySQL, MS SQL)**

* **Зачем нужен вызов метода Class.forName(…)?**

**Это из reflection. Данная конструкция загружает класс в память и если в нем есть блок статической инициализации, то он выполняется. В отношении JDBC – таким образом драйвер регистрирует себя в DriverManager’е.**

* **В чем разница между Statement и PreparedStatement?**

**Statement – обычный строковый запрос, который компилируется непосредственно при выполнении.**

**PreparedStatement – это подготовленный запрос, который характерен тем, что его можно параметризовать через знаки вопроса, на место которых устанавливаются значения, указанные при подготовке запроса. Данный вариант запроса проходит предкомпиляцию. Еще его преимущество в том, что он помогает бороться с SQL Injection.**

* **Что такое ResultSet?**

**Это ответ, который приходит из базы в виде таблицы.**

**Еще вопросы**

* **Что такое Generics и зачем они нужны?**

**Обобщения — это параметризованные типы, они позволяют объявлять классы, интерфейсы и методы, где тип данных, которыми они оперируют, указан в виде параметра. Такие классы, интерфейсы или методы называются обобщёнными классами или обобщёнными методами. Язык Java всегда предлагал возможность создавать в определённой мере обобщённые классы, интерфейсы и методы, оперирующие ссылками на тип Object, поскольку он является суперклассом для всех остальных классов, ссылка на тип Object может обращаться к объекту любого типа. Проблема в том, что они не могут обеспечить безопасность типов. Обобщения добавили в язык безопасность типов и упростили процесс выполнения, поскольку теперь нет необходимости применять явные приведения для транслирования объектов класса Object в реальные типы данных. Благодаря обобщениям, все приведения выполняются автоматически и неявно.**

* **В чем разница между == и equals()?**

**== применяется для сравнения ссылок или примитивных типов**

**equals() применяется для сравнения объектов**

* **Что такое сериализация, десериализация?**

**Сериализация в Java - это операция посредством которой внутреннее состояние объекта переводится в поток байтов. Этот бинарный поток может быть записан на диск, сохранен в памяти, отправлен по сети. Обратная операция - перевод байтов в объект, называется десериализацией.**

* **Что за ключевые слова transient и volatile?**

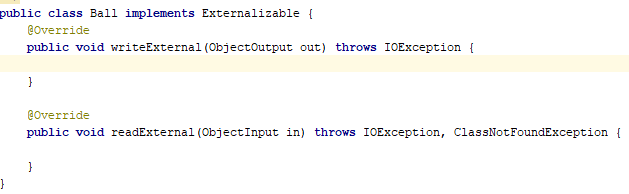
**Transient (нерезидент) — модификатор полей класса в языке Java. Отмеченные этим модификатором поля не записываются в поток байт при применении стандартного алгоритма сериализации. При десериализации объекта такие поля инициализируются значением по умолчанию.**

**Модификатор volatile накладывает некоторые дополнительные условия на чтение/запись переменной. Важно понять две вещи о volatile переменных:**

* **Операции чтения/записи volatile переменной являются атомарными.**
* **Результат операции записи значения в volatile переменную одним потоком, становится виден всем другим потокам, которые используют эту переменную для чтения из нее значения.**
* **В чем разница между Serializable и Externalizable?**

**Если вы говорите, что класс реализует Serializable, то сериализацией управляет сама Java. Т.е. она сама каким-то образом эти объекты разбивает на байты, записывает, читает, восстанавливает**

**Но есть еще один интерфейс Externalizable. Если вы подключите этот интерфейс, то процессом сериализации вы будете управлять вручную.**



**В параметрах методов есть ссылки на входные и выходные потоки, и вы в каждом методе говорите, что вот так я буду писать объект, а вот так читать.**

* **Для запуска Java-программ необходим ли установленный JDK?**

**Нет, только для разработки, для запуска достаточно JRE**

* **Где хранятся объекты?**

**В Java все объекты хранятся в области Heap(Куче). Когда мы определяем переменную с типом класса, создается только ссылка (память для этого не выделяется). Чтобы выделить память для объект, необходимо создать его с помощью new(). Таким образом для этого объекта будет выделена память в области Heap.**

* **Можно ли увеличить размер массива после его инициализации?**

**После создания массива изменить его размер невозможно (хотя можно, конечно, изменять отдельные его элементы). Если в ходе выполнения программы необходимо часто изменять размер массива, лучше использовать другую структуру данных, называемую списком массивов (array list).**

* **Что такое локальный класс?**

**Локальный класс – это класс, объявленный в блоке Java кода. Обычно локальный класс определяется в методе, но он также может быть объявлен в инициализаторе экземпляра класса. Поскольку все блоки Java кода находятся внутри определения класса, то все локальные классы вложены в окружающие классы.**

* **Что такое сборщик мусора (Garbage Collector)?**

**сборка мусора (англ. garbage collection) — одна из форм автоматического управления памятью. Специальный процесс, называемый сборщиком мусора (англ. garbage collector), периодически освобождает память, удаляя объекты, которые уже не будут востребованы приложениями.**

**Garbage Collector должен делать всего две вещи:**

* **Обнаруживать мусор**
* **Очищать память от мусора**
* **Можно ли с помощью Arrays.sort(…) отсортировать массив «наших» объектов?**

**Можно, если «наши» объекты реализуют интерфейс Comparable<T>, а также, реализован алгоритм сравнения в методе compareTo**

* **Что значит, что строки иммутабельны?**

**Это значит, что после создания строки изменить ее нельзя, объект и его поля являются final**

**При попытке изменения строки создается новый объект.**

* **Как создать собственный иммутабельный класс?**

**Иммутабельный класс создается внесением модификаторов final для класса и полей. Поля в иммутабельном объекте обычно инициализируются через конструктор.**

* **Что делает метод toString()?**

**public String toString() – возвращает строковое представление объекта (настолько адекватно, насколько это возможно). В классе Object этот метод реализует выдачу в строку полного имени объекта (с именем пакета), после которого следует символ ‘@’, а затем в шестнадцатеричном виде хэш-код объекта. В большинстве стандартных классов этот метод переопределен. Для числовых классов возвращается строковое представление числа, для строковых – содержимое строки, для символьного – сам символ**

* **В чем разница между набором классов Reader/Writer и InputStream/OutputStream?**

**Reader/Writer - это классы СИМВОЛЬНОГО потока ввода/вывода**

**InputStream/OutputStream – классы БАЙТОВОГО потока ввода/вывода**

* **Что такое области видимости?**

***Это может относиться к модификаторам доступа:***

**В Java существуют следующие модификаторы доступа:**

**private: члены класса доступны только внутри класса;**

**default (package-private) (модификатор, по-умолчанию): члены класса видны внутри пакета (если класс будет так объявлен он будет доступен только внутри пакета);**

**protected: члены класса доступны внутри пакета и в наследниках;**

**public: члены класс доступны всем;**

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

***Или это может относиться к переменным:***

**Когда переменная уже объявлена - это значит, что её можно далее использовать, например, в выражениях или же передавать в функцию, но, чтобы обратиться к переменной нужно знать о так называемой "области видимости" переменной.**

**Эта самая область определяется прежде всего "блоком" - вообще всякий областью, замкнутой в фигурные скобки.**

**В Java область видимости определяется двумя основными способами:**

**- классом**

**- методом**

**Переменная "не видна" коду, если она определена за пределами блока.**

**Области видимости могут быть вложенными.**

