**1.** **Опишите процедуру инициализации полей класса и полей экземпляра класса. Когда инициализируются поля класса, а когда – поля экземпляров класса. Какие значения присваиваются полям по умолчанию? Где еще в классе полям могут быть присвоены начальные значения?**Поля класса инициализируется в момент первой загрузки класса. Поля объекта инициализируются во время конструирования экземпляра класса. По умолчанию ссылочным полям присваивается null, а примитивам особые значения (0, false, 0.0). Начальные значения могут быть так же присвоены в конструкторе или в статическом блоке.

**2. Дайте определение перегрузке методов. Как вы думаете, чем удобна перегрузка методов? Укажите, какие методы могут перегружаться, и какими методами они могут быть перегружены? Можно ли перегрузить методы в базовом и производном классах? Можно ли private метод базового класса перегрузить public методов производного? Можно ли перегрузить конструкторы, и можно ли при перегрузке конструкторов менять атрибуты доступа у конструкторов?**

Перегрузка методов – это объявление в классе методов с одним и тем же именем, но с различными типами и/или количеством параметров. Перегрузка методов удобна тем, что для разных типов объектов мы можем описать общее поведение. Методы можно перегружать как в родительском, так и в дочернем классе.

Private методы не видны за пределами класса, а значит и перегрузить их в производном классе нельзя.

Конструкторы можно перегружать, а также делать private.

**3. Объясните, что такое раннее и позднее связывание? Перегрузка – это раннее или позднее связывание? Объясните правила, которым следует компилятор при разрешении перегрузки; в том числе, если методы перегружаются примитивными типами, между которыми возможно неявное приведение или ссылочными типами, состоящими в иерархической связи.**

Ранее связывание – это когда метод, который будет вызван, известен во время компиляции, например вызов статического метода.

Позднее связывание – это когда метод, который будет вызван, определяется во время компиляции.

Перегрузка – это раннее связывание, а переопределение методов – позднее.

Если есть точное соответствие по типу и количеству параметров, то вызывается именно этот метод. Так же возможно автоматическое расширяющее продвижение примитивных типов (например int до double). И для примитивов, и для объектов компилятор будет стараться вызвать наиболее специфичный метод.

**4. Объясните, как вы понимаете, что такое неявная ссылка this? В каких методах эта ссылка присутствует, а в каких – нет, и почему?**Каждому методу (не статическому) при вызове в качестве параметра передаётся ссылка на объект, в котором находится и вызывается данный метод. Для того, чтобы обратиться к объекту внутри метода используется неявная ссылка this. Начиная с JDK 8 ссылку можно явно передавать в качестве первого параметра метода. Это сделано для возможности аннотировать.

В статических методах нет ссылки this т.к. эти методы принадлежат классу. К нему можно обратиться по имени класса.

**5. Что такое финальные поля, какие поля можно объявить со спецификатором final? Где можно инициализировать финальные поля?**

Финальные поля это по сути константы. Ими могут быть объявлены как поля класса, так и параметры метода и локальные переменные. Финальные поля можно инициализировать либо сразу при объявлении, либо в конструкторе.

**6. Что такое статические поля, статические финальные поля и статические методы. К чему имеют доступ статические методы? Можно ли перегрузить и переопределить статические методы? Наследуются ли статические методы?**

Поля, объявленные с модификатором static, являются общими для всех экземпляров класса. Они инициализируются при первой загрузке класса, как и static блоки.

Static final объявляются константы.

Статические методы, как и статические поля, доступны без обращения к конкретному экземпляру класса. Статические методы могут обращаться только к статическим переменным и методам.

Статические методы можно перегружать, но нельзя переопределять. Фактически статические методы наследуются.

**7. Что такое логические и статические блоки инициализации? Сколько их может быть в классе, в каком порядке они могут быть размещены и в каком порядке вызываются?**

Статический блок инициализации:

static {  
 int a = 10;  
}

Логический блок инициализации:

{  
 int a = 10;  
}

В блоках инициализации происходит инициализация переменных (присвоение им начальных значений). Кол-во блоков в классе не ограничено. Размещены могут быть в любом порядке.

Статические блоки инициализации вызываются всего 1 раз при первой загрузке класса.

Логические блоки инициализации вызываются каждый раз при создании нового экземпляра класса.

**8. Что представляют собой методы с переменным числом параметров, как передаются параметры в такие методы и что представляет собой такой параметр в методе? Как осуществляется выбор подходящего метода, при использовании перегрузки для методов с переменным числом параметров?**

Это методы, которые принимают переменное число параметров. В итоге аргумент является неявным массивов, с которым можно обращаться как с массивом. Параметр переменной длины должен быть указан последним среди всех параметров метода. Метод может иметь только 1 параметр с переменным количеством аргументов.

1. Если мы используем перегрузку с разными типами (int … a) и (double … a) то выбирается метод исходя из типа передаваемых параметров.
2. Второй способ перегрузки заключается в том, чтобы добавить 1 или несколько обычных параметров. (int … a) (Boolean b, int … a). Тогда выбирается подходящий вариант.
3. Перегрузка методом, который не содержит аргумент переменной длины (int … a) и (int a).
4. Иногда возникают неоднозначности:

**static** **void** **vaTest** (**int** ... v) { // ...

**static** **void** **vaTest** (**int** n, **int** ... v) { // ...

компилятор не сможет разрешить следующий вызов vaTest(l);

**static** **void** **vaTest** (**int** ... v) { // ...

**static** **void** **vaTest** (**boolean** ... v) { // ...

Компилятор не сможет разрешить вызов vaTest();

**9. Чем является класс Object? Перечислите известные вам методы класса Object, укажите их назначение.**

Все классы являются производными от класса Object. => класс Object является суперклассом для всех классов, и ссылочная переменная из класса Object может ссылаться на объект любого другого класса. А поскольку массивы реализованы в виде классов, то ссылочная переменная типа Object может ссылаться и на любой массив.

У класса есть несколько важных методов.

* Object **clone**() - создаёт новый объект, не отличающий от клонируемого
* **boolean** **equals**(Object obj) - определяет, равен ли один объект другому
* **void** **finalize**()- вызывается перед удалением неиспользуемого объекта
* Class<?> getClass()- получает класс объекта во время выполнения
* **int** **hashCode**()- возвращает хеш-код, связанный с вызывающим объектом
* **void** **notify**()- возобновляет выполнение потока, который ожидает вызывающего объекта
* **void** **notifyAll**()- возобновляет выполнение всех потоков, которые ожидают вызывающего объекта
* String **toString**()- возвращает строку, описывающий объект
* **void** **wait**()- ожидает другого потока выполнения
* **void** **wait**(**long** millis)- ожидает другого потока выполнения
* **void** **wait**(**long** millis, **int** nanos)- ожидает другого потока выполнения

Методы **getClass()**, **notify()**, **notifyAll()**, **wait()** являются финальными и их нельзя переопределять.

**10. Что такое хэш-значение? Объясните, почему два разных объекта могут сгенерировать одинаковые хэш-коды?**

Хэш код – это целое значение, которое возвращает метод hashCode(). По умолчанию – это случайное число. Но переопределив метод, можно создать хеш-код, зависящий от переменных объекта. Именно поэтому, разные объекты могут иметь одинаковый хэш-код.

**11. Что такое объект класса Class? Чем использование метода getClass() и последующего сравнения возвращенного значения с Type.class отличается от использования оператора instanceof?**

Класс с именем Class представляет характеристики класса (что-то вроде метаданных объекта), экземпляром которого является объект.

getClass() или class-literal - Foo.class возвращают объект Class, который содержит некоторые метаданные о классе:

1. название
2. пакет
3. методы
4. поля
5. конструкторы
6. аннотации

В классе Class нет конструкторов, экземпляр этого класса создается исполняющей системой Java во время загрузки класса и предоставляется методом getClass() класса Object

При сравнении двух объектов при помощи instanceOf будет возвращено true для подклассов, что не верно для использования в методе equals(). При использовании метода getClass и последующего сравнения будет возвращен true только для объектов того же класса.

**12. Укажите правила переопределения методов** **equals(), hashCode() и toString().**

equals(), hashCode() желательно переопределять для всех классов.

При переопределении метода equals разработчик должен придерживаться основных правил, определенных в спецификации языка Java.

* **Рефлексивность**

для любого заданного значения x, выражение x.equals(x) должно возвращать true.  
*Заданного* — имеется в виду такого, что x != null

* **Симметричность**

для любых заданных значений x и y, x.equals(y) должно возвращать true только в том случае, когда y.equals(x) возвращает true.

* **Транзитивность**

для любых заданных значений x, y и z, если x.equals(y) возвращает true и y.equals(z) возвращает true, x.equals(z) должно вернуть значение true.

* **Согласованность**

для любых заданных значений x и y повторный вызов x.equals(y) будет возвращать значение предыдущего вызова этого метода при условии, что поля, используемые для сравнения этих двух объектов, не изменялись между вызовами.

* **Сравнение null**

для любого заданного значения x вызов x.equals(null) должен возвращать false.

### Общий алгоритм определения equals

1. Проверить на равенство ссылки объектов this и параметра метода o.  
   if (this == o) return true;
2. Проверить, определена ли ссылка o, т. е. является ли она null.  
   Если в дальнейшем при сравнении типов объектов будет использоваться оператор instanceof, этот пункт можно пропустить, т. к. этот параметр возвращает false в данном случае null instanceof Object.
3. Сравнить типы объектов this и o с помощью оператора instanceof или метода getClass(), руководствуясь описанием выше и собственным чутьем.
4. Если метод equals переопределяется в подклассе, не забудьте сделать вызов super.equals(o)
5. Выполнить преобразование типа параметра o к требуемому классу.
6. Выполнить сравнение всех значимых полей объектов:
   * для примитивных типов (кроме float и double), используя оператор ==
   * для ссылочных полей необходимо вызвать их метод equals
   * для массивов можно воспользоваться перебором по циклу, либо методом Arrays.equals()
   * для типов float и double необходимо использовать методы сравнения соответствующих оберточных классов Float.compare() и Double.compare()
7. И, наконец, ответить на три вопроса: является ли реализованный метод симметричным? Транзитивным? Согласованным? Два других принципа (рефлексивность и определенность), как правило, выполняются автоматически.

### Контракт hashCode

Для реализации хэш-функции в спецификации языка определены следующие правила:

* вызов метода hashCode один и более раз над одним и тем же объектом должен возвращать одно и то же хэш-значение, при условии что поля объекта, участвующие в вычислении значения, не изменялись.
* вызов метода hashCode над двумя объектами должен всегда возвращать одно и то же число, если эти объекты равны (вызов метода equals для этих объектов возвращает true).
* вызов метода hashCode над двумя неравными между собой объектами должен возвращать разные хэш-значения. Хотя это требование и не является обязательным, следует учитывать, что его выполнение положительно повлияет на производительность работы хэш-таблиц.