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, принадлежащие классу, являются статическими, а поля, принадлежащие объекту, - нестатическими. Статические поля доступны без создания объекта класса. Соответственно инициализироваться статические и нестатические поля должны в разное время: одни до создания объекта класса, а другие после. Поля класса инициализируются при загрузке класса. Класс загружается при первом обращении к нему в программе. Нестатические поля экземпляра класса инициализируются при вызове конструктора создаются при создании объекта, которому они принадлежат. Поля по умолчанию принимают значения зависящие от их от объявленного им типа т.е 0, 0.0, “ ”, false для примитивов и null для ссылочных типов.

Способы инициализации - Инициализация в месте объявления поля, в ициализационном блоке и в конструкторе для экземпляров класса.

1. **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 возможность создавать несколько методов с одинаковым названием, но разными параметрами. Одним из главных плюсов перегрузки является то, что уменьшается номенклатура названий методов**,** выполняющих похожую функцию. Статические методы могут быть перегружены нестатическими и наоборот. Методы могут быть перегружены как в базовом так ив производном классах (при условия возможности видимости методов родителя наследником). Private метод базового класса можно перегрузить только в самом базовом классе (атрибуты доступа). Конструкторов может быть несколько в классе. В этом случае конструкторы называют перегруженными. При перегрузке конструктора модификатор доступа можно менять (но все зависит от видимости класса).

1. **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.**

* -Существует два типа связывания методов в языке Java: ранее связывание (его ещё называют статическим) и позднее (соответственно, динамическое) связывание. Т.е. 1 - этозначит, что метод при вызове привязывается к конкретному коду в момент компиляции, а второе - во время выполнения, при запуске программы и создании объектов. Перегрузка это раннее связывание. Выбор метода во время компиляции происходит на основе типа ссылочной переменной и на основе порядка следования аргументов. Java будет использовать автоматическое преобразование типов только при отсутствии полного соответствия. Т.е. если нету метода с аргументами int, а есть с аргументами double то вызовется он.

1. **Объясните, как вы понимаете, что такое неявная ссылка this? В каких методах эта ссылка присутствует, а в каких – нет, и почему??**

* Это ссылка на текущий объект. Компилятор видит все поля и методы объекта через эту ссылку. т.е. это ссылка на поле/ метод экземпляра класса который вызвал метод. Ссылка this может быть использована только для нестатического метода. Статические методы нельзя вызвать на объекте, их смысл быть глобальными. Т.е. в статический метод не передается ссылка this.

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

* - Финальные поля (константы)– поля которые можно инициализировать только раз. Любые необходимые поля можно объявить final. Константа может быть объявлена в поле класса, но не проинициализирована. В этом случае она должна быть проинициализирована в логическом блоке класса, заключенном в {}, или конструкторе.

1. **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?**

* Cтатические поля являются общими для всего класса и могут использоваться без создания объектов класса. Статические финальные поля являются константами для всего класса. (см. пункт 5). Статические методы также относятся ко всему классу в целом т.е. когда методы не затрагивают состояние объекта, то есть его нестатические поля и константы, и для вызова метода нет смысла создавать экземпляр класса. В статических методах мы можем вызывать только другие статические методы и использовать только статические переменные. Статический метод может быть перегружен но не переопределен. Статический метод наследуется в подклассе, но это не полиморфизм. При написании реализации статического метода родительский метод класса является скрытым, а не переопределенным.

1. **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?**

* блоки инициализации используются для инициализации переменных внутри класса;

есть два типа блоков инициализации - статический и нестатический;

синтаксис очень простой - просто пишем выражение внутри блока из двух скобок:

статический блок инициализации используется для статических переменных, нестатический - для всех остальных;

блоки инициализации делают код читабельнее, и позволяют вызывать любые методы.

В классе может быть множство блоков инициализации. Порядок их записи может быть любой т.е. в любом месте. Порядок выполнения следующий:

- Статические блоки super класса. (все в порядке написания).

- статические блоки класса (все в порядке написания).

- нестатические блоки класса (все в порядке написания).

- конструктор.

1. **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?**

* В языке Java существуют методы, которые могут принимать переменное количество аргументов. Они называются методами с аргументами переменной длины.

static void varTest(int ... v) { //}

Набор объектов, переданный в такой метод, преобразуется в массив с типом и именем, которые указаны в качестве параметров метода.

Метод, который получает аргументы переменной длины, может иметь несколько перегруженных реализаций. Такие реализации отличаются между собой типом получаемых значений т.е. сразу будет найден метод с жестко фиксированным числом параметров, потом, параметризованный метод и только потом выбран метод с переменным числом параметров.

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

* В Java есть специальный суперкласс Object и все классы являются его подклассами. Поэтому ссылочная переменная класса Object может ссылаться на объект любого другого класса. Так как массивы являются тоже классами, то переменная класса 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
* 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

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

* Хеш-код - это целое число, генерируемое на основе конкретного объекта. Его можно рассматривать как шифр с уникальным значением. Функция hashCode() объекта Object возвращает целое число int, размер которого равен 4-м байтам. По умолчанию, функция hashCode() для объекта возвращает номер ячейки памяти, где объект сохраняется. Поэтому, если изменение в код приложения не вносятся, то функция должна выдавать одно и то же значение. При незначительном изменении кода значение hashCode также изменится.
* Объекты могут быть не эквивалентны но иметь одинаковый хэш-код. Существует большое количество объетов с очень большим разбросом вероятностей по значению, например класс обертка Long. Так как метод hashCode() возвращает int, то для большого количества значений Long хэш-код будет идентичный при неэквивалентности самих объектов.

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

* - Объект класс Class создается при закрузке класса и возвращает ссылку на метаобъект типа класс. С его помощью можно получать информацию о классе, к которому принадлежит объект. Оператор instanceof нужен, чтобы проверить, был ли объект, на который ссылается переменная создан на основе какого-либо класса

**-** Function fun = new Function();  
System.*out*.println(fun instanceof Object); // true  
System.*out*.println(fun.getClass()); // class com.androsor.string.Fraction

* (проверка с помощью instanceof будет возвращать true для подклассов и будет работать правильно только в случае если ваш класс объявлен как immutable).

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

* - Рефлексивность: Объект должен равняться себе самому.
* Симметричность: если a.equals(b) возвращает true, то b.equals(a) должен тоже вернуть true.
* Транзитивность: если a.equals(b) возвращает true и b.equals(c) тоже возвращает true, то c.equals(a) тоже должен возвращать true.
* Согласованность: повторный вызов метода equals() должен возвращать одно и тоже значение до тех пор, пока какое-либо значение свойств объекта не будет изменено. То есть, если два объекта равны в Java, то они будут равны пока их свойства остаются неизменными.
* Сравнение null: объект должны быть проверен на null. Если объект равен null, то метод должен вернуть false, а не NullPointerException. Например, a.equals(null) должен вернуть false.
* Если объекты равны по результатам выполнения метода equals, тогда их hashcode должны быть одинаковыми.
* Если объекты не равны по результатам выполнения метода equals, тогда их hashcode могут быть как одинаковыми, так и разными. Однако для повышения производительности, лучше, чтобы разные объекты возвращали разные коды.