***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?***

Полей класса – при создании или в статическом блоке.

Полей экземпляра – при создании, блоке инициализации, в конструкторе.

По умолчанию:

byte = 0

short = 0

int = 0

long = 0

float = 0.0

double = 0.0

char = '\u0000'

boolean = false

Object = null

***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 возможность создавать несколько методов с одинаковым названием, но разными параметрами.

Можно ли private метод базового класса перегрузить public методов производного. Нет, т.к. его не вызвать из вне.

***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.***

Раннее и позднее связывание(статическая и динамическая типизация)

Можно понять из названия, статическое связывание носит более статический характер, так как происходит во время компиляции, то есть код «знает», какой метод вызывать после компиляции исходного кода на Java в файлы классов. А поскольку это относится к ранней стадии жизненного цикла программы, то называется также ранним связыванием (early binding).

С другой стороны, динамическое связывание происходит во время выполнения, после запуска программы виртуальной машиной Java. В этом случае то, какой метод вызвать, определяется конкретным объектом, так что в момент компиляции информация недоступна, ведь объекты создаются во время выполнения. А поскольку это происходит на поздней стадии жизненного цикла программы, то называется в языке Java поздним связыванием (late binding).

Статическое связывание используется в языке Java для разрешения перегруженных методов, в то время как динамическое связывание используется в языке Java для разрешения переопределенных методов.

class Main {

static void method1(int x) {

System.out.println("1");

}

static void method1(long x) {

System.out.println("2");

}

static void method1(float x) {

System.out.println("3");

}

static void method1(double x) {

System.out.println("4");

}

static void method1(Short x) {

System.out.println("5");

}

static void method1(short... x) {

System.out.println("6");

}

public static void main(String[] args) {

short x = 20;

method1(x);

}

}

Ответ: 1

Выбор перегруженного метода происходит по следующим правилам:

1. Выбирается наиболее подходящий метод. Если бы у нас был метод, принимающий short x , то он бы и выполнился. Если нужно, то может выполняться расширяющее преобразование примитивов. Выбирается метод, для которого нужно наиболее близкое расширяющее преобразование. В данном случае до int , так как long , float и double «большие» типы.

2. Расширяющее преобразование примитивов имеет приоритет над автоупаковкой/распаковкой.

3. Автоупаковка/распаковка имеет приоритет перед методом с переменным количеством аргументов.

4. Сужающее преобразование примитивов автоматически НЕ выполняется.

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

Внутри метода экземпляра или конструктора this это ссылка на текущий объект - объект, метод или конструктор которого вызывается. Вы можете ссылаться на любой член текущего объекта из метода экземпляра или конструктора, используя this.

Не присутствует в static потому что объекта не существует.

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

Финальное поле = это поле, которое не может поменять значение.  
Финальный класс = это класс без наследников.

Финальный метод = это метод, который нельзя переопределить.

Инициализация финальных полей:

1. инициализировать тогда, когда она объявлена.

2. может быть инициализирована внутри блока инициализатора экземпляра или внутри конструктора. Если у вас есть более одного конструктора в вашем классе, то он должен быть инициализирован во всех из них, иначе будет вызвана ошибка времени компиляции.

3. может быть инициализирована внутри статического блока.

***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?***

Статические поля, это такие поля к которым можно обращаться без создания объекта класса. static – переменная одна для всех объектов класса. Вызывается через ИмяКласса.поле, а не через объект.

Финальные статические поля = константы.

Статический метод:

1. Метод вызывается без создания объекта класса.

2. Статические методы нельзя переопределять.

3. Статическим методам нужен "статический контекст".

Статическое поле или метод = "один для класса и его наследников" = не наследуется

***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?***

Блоки инициализации представляют собой код, заключенный в фигурные скобки и размещаемый внутри класса вне объявления методов или конструкторов.

Существуют статические и нестатические блоки инициализации.

Блок инициализации выполняется перед инициализацией класса загрузчиком классов или созданием объекта класса с помощью конструктора.

Несколько блоков инициализации выполняются в порядке следования в коде класса.

Блок инициализации способен генерировать исключения, если их объявления перечислены в throws всех конструкторов класса.

Блок инициализации возможно создать и в анонимном классе.

Статические блоки инициализация используются для выполнения кода, который должен выполняться один раз при инициализации класса загрузчиком классов, в момент, предшествующий созданию объектов этого класса при помощи конструктора. Такой блок (в отличие от нестатических, принадлежащих конкретному объекту класса) принадлежит только самому классу (объекту метакласса Class).

***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?***

В последний параметр всегда массив вида array… Не может быть более одного массива с переменным числом.

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

Object - это корень иерархии классов. Каждый класс является потомком для Object. Все объекты, включая массивы, реализуют методы этого класса.

protected Object clone() Создает и возвращает копию этого объекта.

boolean equals(Object obj) Указывает, равен ли какой-либо другой объект этому.

protected void finalize() Вызывается сборщиком мусора для объекта, когда сборщик мусора определяет, что на этот объект больше нет ссылок.

Class<?> getClass() Возвращает runtime class этого объекта.

int hashCode() Возвращает значение хэш-кода для объекта.

void notify( ) Пробуждает единственный поток

void notifyAll() Пробуждает все потоки

String toString() Возвращает строковое представление объекта.

void wait() Заставляет текущий поток ждать, пока другой поток не вызовет метод notify() или метод notifyAll () для этого объекта.

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

Хэш-код - целочисленное значение, связанное с каждым объектом в Java. Его основная цель - облегчить хеширование в хеш-таблицах, которые используются структурами данных, такими как HashMap.

Два разных объекта могут сгенерировать одинаковый хэш-код изза того что:

1. криво написан метод для этих объектов, например return 1;

2. так случайно получилось, что при корректной реализации метода и разных значениях в объектах, выдало одинаковое число

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

Class object, which contains some metadata about the class: name, package, methods, fields, constructors, annotations.

instanceof tests whether the object reference on the left-hand side is an instance of the type(экземпляром типа) on the right-hand side or some subtype.

getClass() == ... tests whether the types are identical.

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

Для equals():

1. Рефлексивность: Объект должен равняться себе самому.

2. Симметричность: если a.equals(b) возвращает true, то b.equals(a) должен тоже вернуть true.

3. Транзитивность: если a.equals(b) возвращает true и b.equals(c) тоже возвращает true, то c.equals(a) тоже должен возвращать true.

4. 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.

5. 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.

Для hashCode()

1. Если объекты равны по результатам выполнения метода equals, тогда их hashcode должны быть одинаковыми.

2. 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.