**Programming with classes**

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?**

Первый способ инициализации – это присвоить значения переменным при их создании.

Class Student{

String name = “Bill”; int age = 20;}

Второй способ – инициализировать поля через конструктор класса.

Class Student{

String name ;

int age ;

public Student(String name, int age) {

this.name = name;

this.age = age;

}

}

Поля объекта инициализируются во время конструирования экземпляра класса. Инициализация полей класса произойдет во время загрузки класса.

По умолчанию объектам присваивается null, цифровым типам 0, значениям типа boolean - false, char = ' ' переменной типа char присваивается нуль-символ, который отображается в виде символа «пробел».

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 создание нескольких версий одного метода с одним и тем же именем, но разным списком параметров.

Перегрузка методов удобна, тем что не нужно создавать новых методов, для работы с разным набором входных параметров.

Методы могут перегружаться методами с различными по типу и количеству параметрами. Статические методы могут перегружаться нестатическими методами, и наоборот.

Методы с одинаковыми именами, но с различным списком параметров и возвращаемыми значениями могут находиться в разных классах одной цепочки наследования и также будут перегруженными.

Конструкторы поддерживают перегрузку, как и методы. В конструкторе допускается применение модификаторов доступа, можно задать один из модификаторов: **public,** **protected, private** или без модификатора.

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.**

Раннее связывание (статическое) происходит во время компиляции, т.е. код знает, какой метод вызвать после компиляции исходного кода на Java в файлы классов.

Позднее связывание (динамическое) 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.

Перегрузка реализует раннее связывание, т.е. версия вызываемого метода определяется на этапе компиляции.

При перегрузке примитивными типами если у нас будет метод, принимающий значения типа short, то он и выполнится если мы передаем тип short. Если нет метода с такими входными данными, то выполняется расширяющее преобразование примитивов. Выбирается метод, для которого нужно наиболее близкое расширяющее преобразование. В данном случае, например, до int.

Расширяющее преобразование примитивов имеет приоритет над автоупаковкой/автораспаковкой.

Автоупаковка/автораспаковка имеет приоритет перед методом с переменным количеством аргументов.

Сужающее преобразование примитивов автоматически не выполняется.

Если перегрузка осуществляется ссылочными типами, состоящими в иерархической связи.

class Main {

    static class Monster {}

    static class Goblin extends Monster {}

    static class Hobgoblin extends Goblin {}

    static void method1(Monster obj) {

        System.out.println("1");

    }

    static void method1(Goblin obj) {

        System.out.println("2");

    }

     public static void main(String[] args) {

        Monster obj = new Hobgoblin();

        method1(obj);

    }

}

В консоль будет выведено 1. Так как переменная obj имеет тип Monster. Выбор перегруженного метода осуществляется на этапе компиляции, а не на этапе выполнения, поэтому метод выбирается на основе типа объявленной переменной.

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

Каждый экземпляр класса (объект) имеет неявную ссылку this на себя, которая передается неявно и нестатическим методам класса. После этого каждый метод знает, какой объект его вызвал. Если мы обращаемся к атрибуту, например, с именем id в методах можно писать this.id, но это не обязательно, так как записи id и this.id будут равносильны. Но если в методе есть локальная переменная или параметр с таким же именем, как и поле класса, то для обращения к полю класса необходимо указывать this.id. Без указания обращение будет производиться к локальной переменной.

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

Модификатор final используется для определения констант в качестве члена класса, локальной переменной или параметра метода. Методы, объявленные как final, нельзя замещать в подклассах. Для классов, объявленных со спецификатором final, нельзя создавать подклассы.

Константа может быть объявлена как поле класса, но не проинициализирована. В этом случае она должна быть проинициализирована в логическом блоке класса, заключенном в {}, или конструкторе, но только в одном из указанных мест. Значение по умолчанию константа получить не может в отличие от переменных класса.

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?**

Поле данных, объявленное в классе как static, является общим для всех объектов класса и называется переменной класса.

Для работы со статическими атрибутами используются статические методы, объявленные со спецификатором static.

По причине недоступности указателя this статические поля и методы не могут обращаться к нестатическим полям и методам напрямую, так как они не «знают», к какому объекту относятся, да и сам экземпляр класса может быть не создан. Для обращения к статическим полям и методам достаточно имени класса, в котором они определены.

Переопределение статических методов невозможно, так как обращение к статическому атрибуту или методу осуществляется посредством задания имени класса, которому они принадлежат.

Статические методы могут быть перегружены так же, как и любой другой метод.

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?**

Логическим блоком называется код, заключенный в фигурные скобки и не принадлежащий ни одному методу текущего класса, например:

{ */\* код \*/* }

**static** { */\* код \*/* }

Логические блоки чаще всего используются в качестве инициализаторов полей, но могут содержать вызовы методов и обращения к полям текущего класса. При создании объекта класса они вызываются последовательно, в порядке размещения, вместе с инициализацией полей как простая последовательность операторов, и только после выполнения последнего блока будет вызван конструктор класса. Операции с полями класса внутри логического блока до явного объявления этого поля возможны только при использовании ссылки **this**, представляющую собой ссылку на текущий объект.

Статический блок инициализации служит для инициализации статических переменных.

static {

   //...

}

Статический блок — это, в сущности, конструктор для всего класса. Он ставится между определениями полей и функциями класса. Команды будут выполняться в одном из двух случаев, том, который наступит раньше:

При создании первого объекта класса в процессе работы программы, перед запуском конструктора.

При первом вызове статической функции, перед выполнением.

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?**

Методы с переменным числом параметров принимают в метод нефиксированное число параметров, что позволяет отказаться от создания сложного объекта для последующей передачи его в метод.

Список параметров в методе выглядит следующим образом (Тип… args). А в случае необходимости передачи параметров других типов:

(Тип1 t1, Тип2 t2, Тип… args)

Методы с переменным числом аргументов могут быть перегружены:

void methodName(Integer...args) { }

void methodName(int x1, int x2) { }

Такие реализации отличаются между собой типом получаемых значений.

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

Класс Object является суперклассом для всех классов. В классе Object определен набор методов, которые наследуются всеми классами:

**protected Object clone()** — создает и возвращает копию вызывающего объекта; **public  boolean  equals(Object  ob)**  —  предназначен для использования и пере определения в подклассах с выполнением общих соглашений о сравнении содержимого двух объектов одного и того же типа; **public Class<? extends Object> getClass()** — возвращает объект типа Class; **protected void finalize()** — автоматически вызывается сборщиком мусора (garbage collection) перед уничтожением объекта; **public int hashCode()**— вычисляет и возвращает хэш-код объекта (число, в общем случае вычисляемое на основе значений полей объекта); **public String toString()** — возвращает представление объекта в виде строки.

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

В классе Object есть метод int hashCode(), его можно переопределить и как правило в каждом классе он возвращает число, являющееся уникальным идентификатором объекта.

Все одинаковые по содержанию объекты имеют одного типа должны иметь одинаковые хэш-коды.

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

Метод **getClass()** возвращает класс объекта, содержащий сведения об объекте: public final Class<?> getClass().

Используя оператор instanceof, можно узнать, от какого класса произошел объект. Этот оператор имеет два аргумента. Слева указывается ссылка на объект, а справа — имя типа, на совместимость с которым проверяется объект. Оператор instanceof опирается не на тип ссылки, а на свойства объекта, на который она ссылается. Но этот оператор возвращает истинное значение не только для того типа, от которого был порожден объект.

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

Java предлагает следующие правила для переопределения методов:

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.

Метод hashCode() возвращает хэшкод объекта, вычисление которого управляется следующими соглашениями:

• все одинаковые по содержанию объекты одного типа должны иметь одинаковые хэш-коды;

• различные по содержанию объекты одного типа могут иметь различные хэш-коды;

• во время работы приложения значение хэш-кода объекта не изменяется, если объект не был изменен.

Метод toString() следует переопределять таким образом, чтобы, кроме стандартной информации о пакете в котором находится класс, и самого имени класса, он возвращал значения полей объекта, вызвавшего этот метод (т. е. всю полезную информацию объекта), вместо хэшкода, как это делается в классе Object.