Сlasses and Objects

**1. Дайте определение таким понятиям как ”класс” и “объект”. Приведите примеры объявления класса и создания объекта класса. Какие спецификаторы можно использовать при объявлении класса.**

**2. Как вы определяете, какие поля и методы необходимо определить в классе, приведите пример. Какие спецификаторы можно использовать с полями, а какие с методами (и что они значат)?**

|  |  |  |  |
| --- | --- | --- | --- |
| **Модификатор** | **Класс** | **Поле класса** | **Метод** |
| **Доступа** | **+** | **+** | **+** |
| **Static** | **+** | **+** | **+** |
| **final** | **+** | **+** | **+** |
| **synchronized** | **-** | **-** | **+** |
| **transient** | **-** | **+** | **-** |
| **volatile** | **-** | **+** | **-** |
| **Abstract** | **+** | **-** | **+** |

Public class Cat {}

Private static class Cat{} – создание классов.

New Cat(); - создание объекта класса Cat.

**3. Что такое конструктор? Как вы отличите конструктор от любого другого метода? Сколько конструкторов может быть в классе? Что такое конструктор по умолчанию, может ли в классе совсем не быть конструкторов? Объясните, что делает оператор this() в конструкторе?**

Конструктор - это специальный метод, который вызывается при создании нового объекта.

Конструкторов может быть сколько угодно. К-р по умолчанию = конструктору без параметров. Если переопределен другой конструктор, то по умолчанию не работает, нужно указывать явно. Не быть конструктора не может, но может быть приватный (паттерн синглтон)

This –означает обращаение к этому объекту, используется в след случаях:

Public User(String name, int age) {

This.name = name;

This.age = age;

}

**4. 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 инициализуруются при запуске JVM, если поле – статическое; и при создании экземпляра класса, если поле не статическое.

Каждый тип имеет свое значение по умолчанию (Boolean = false, int=0; object=null и т.д.). Значение по умолчанию устанавливается при инициализации.

Статические переменные хранятся в особом блоке non-heap memory – metaspace.

Обычные переменные экземпляров класса хранятся в heap.

Иным способом можно инициализировать static блоками, которые вызываются перед вызовом даже конструктора класса.

**5. 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?**

В программе мы можем использовать методы с одним и тем же именем, но с разными типами и/или количеством параметров. Такой механизм называется **перегрузкой методов** (method overloading).

Стоит отметить, что на перегрузку методов влияет количество и типы параметров, но тип возвращаемого значения не играет никакой роли.

Перегружать и переопределять методы можно и в базовом и в производном классе. В производном можно перегружать/определять методы базового класса.

Перегружать конструктора также можно, менять артибуты доступа тоже можно.

**6. 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.**

Присоединение вызова метода к телу метода называется связыванием

*Если связывание проводится перед запуском программы (компилятором и компоновщиком, если он есть), оно называется ранним связыванием (*early binding*).*

*Проблема определения метод какого объекта вызывать решается благодаря позднему связыванию (*late binding*), то есть связыванию, проводимому во время выполнения программы, в зависимости от типа объекта. Позднее связывание также называют динамическим (*dynamic binding*) или связыванием на стадии выполнения (*runtime binding*).*

Раннее связывание (его еще называют статическим) – происходит рано (на этапе компиляции) программы, то есть мы заранее знаем какой метод будет вызван.

Позднее – динамическое, будет вызвано в рантайме, то есть заранее точно неизвестно какой метод будет вызван.

Перегрузка – это раннее связывание.

Ссылка на пример <http://www.cyberforum.ru/java-j2se/thread1464117.html>

**class** B **implements** A {

**public** **void** callback() {

**System**.out.println("Реализация интерфейса в класса B");

    }

}

**class** C **implements** A {

**public** **void** callback() {

**System**.out.println("Реализация интерфейса в класса C");

    }

}

|  |  |
| --- | --- |
| 2  3  4  5  6  7 | A ob;         *// переменная типа ссылки на интерфейс A*    ob = **new** B(); *// ссылка на объект присваивается переменной типа ссылки на интерфейс*  ob.callback(); *// вызов callback из класса B*    ob = **new** C();  ob.callback(); *// вызов callback из класса C* |

Это будет поздним связыванием,т.к. компилятор не знает какой объект будет создан. Будет знать только JVM в процессе программы.

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

Неявная ссылка this – имеется ввиду неявно подставленная компилятором. Например:

**public class** A {  
 **int m**;  
   
 **int** m(){  
 *//return this.m;* **return m**;  
 }  
}

Здесь в коде написано две идентичные строки. В первом случае this можно указать, но не обязательно, компилятор поставит сам. Иногда нужно ставить this явно: может использоваться в контрукторах для решения конфликта имен или в конструкторах для вызова другого конструктора.

Пример второго (первый объяснить любым простейшим примером):

Public Book(int age, int price){

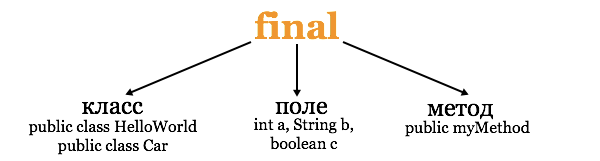
This(age);

This.price = price;

}

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

Поля с final – это константы. Дальнейшее изменение final полей невозможно. Вы можете применять этот модификатор **тремя способами**: для класса, для поля (переменной) и для метода.



Поле можно сделать финальным, тогда оно будет константой. Константы лучше объявлять форматом ***CUSTOMER\_ID***, а не камел кейс!

Можно инициализировать и объявить файнл переменную сразу, а можно объявить ее, и тогда значение можно будет присвоить **один** раз.

**9. 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 – поля с модификатором static. Статические поля являются общими для всего класса, поэтому их можно использовать без создания экземпляра класса.

Статическое финальное поле = константа.

Статические методы также являются общими и не требуют создания экземпляра класса. Для доступа к ним достаточно имя класса.

Переопределить статические методы **нельзя**! Но есть механизм скрытия.

Статические методы не могут быть переопределены в точном смысле слова, но они могут скрыть родительские статические методы

При использовании статических методов надо учитывать ограничения: в статических методах мы можем вызывать только другие статические методы и использовать только статические переменные.

**Скрытие:** Статические методы родительского класса не являются частью дочернего класса (хотя они доступны), поэтому не стоит переопределять его. Даже если вы добавите другой статический метод в подкласс, идентичный методу в его родительском классе, этот статический метод подкласса является уникальным и отличается от статического метода в его родительском классе.

**10. 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?**

Вызываются в порядке сверху вниз. Сначала идут static блоки, после – перед конструктором вызываются логические блоки. Статик переменные также объявляются на том же этапе что и статик блоки (java неявно ставит эти блоки сама при запуске для инициализации статик полей).

**11. 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?**

<https://vscode.ru/prog-lessons/peremennoe-chislo-argumentov-java.html>

Для того, чтобы задать **переменное число аргументов** в методе на языке Java используется оператор многоточие (**…**).

Работа с методом переменной длинны идет аналогично работе с массивом. Та же нумерация, те же методы и т.д. Метод переменной длинны в методе может быть только один и должен находится на последнем месте (если идет вместе с обычными аргументами).

Его можно перегружать.

Выбор осуществляется также как и с обычной перегрузкой. Если невозможно будет определить какой метод вызвать будет ошибка. Компиляции. Например если есть несколько методов, которые могут принять 0 параметров, а мы вызываем method();