**4. 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?

Answer

Объявляются поля следующим образом:

[access\_modifier] [static] [final] type name [= initial value];

*Квадратные скобки [] вокруг некоторых ключевых слов означают, что этот параметр является необязательным. Требуется только тип и имя.*

1. Поле класса – это переменная класса с ключевым словом ‘static’.

Таким образом, независимо от того, сколько объектов вы создаете для этого класса, будет существовать только одно поле, расположенное в классе, и значение этого поля одинаково, независимо от того, к какому объекту он доступен.

1. Поле экземпляра класса – это переменная экземпляра класса. Каждый экземпляр может иметь свои собственные значения для этих полей.

Способы инициализации полей класса:

1. инициализация поля сразу во время объявления:

**class** Cat {

String name = "Том";

**int** age = 2;

}

1. инициализация поля через конструктор:

**class** Cat {

String name;

**int** age;

**public** Cat(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

}

1. инициализация с помощью секции инициализации

**public** **class** CDataInit{

**int** a, b, c, d;

{

a = 1;

b = 2;

c = 3;

d = 4;

}

}

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?

Answer

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

Перегрузка методов даёт возможность создавать методы с одним названием, но с разными параметрами.

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.

Answer

Связывание означает наличие связи между ссылкой и кодом. Например, переменная, на которую вы ссылаетесь, привязана к коду, в котором она определена. Аналогично, вызываемый метод привязан к месту в коде, где он определен.

***Раннее связывание*** (его ещё называют ***статическим***) – 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). Предназначено для ***перегруженных методов***.

***Позднее связывание*** (его ещё называют ***динамическим***) – 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). Предназначено для ***переопределенных методов.***

Перегрузка – это раннее связывание, то есть версия вызываемого метода определяется на этапе компиляции.

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

Answer

***This*** – это ссылка на поле или конструктор экземпляра класса.

Как правило, применять this нужно **в двух случаях:**

1. Когда у переменной экземпляра класса и переменной метода/конструктора одинаковые имена;
2. Когда нужно вызвать конструктор одного типа (например, конструктор по умолчанию или параметризированный) из другого. Это еще называется явным вызовом конструктора.

|  |  |
| --- | --- |
| A | B |
| Когда мы ходим обратиться к переменной экземпляра мы пишем ‘***this .*** *название переменной’.* Если этого не сделать, то значение будет браться из ближайшей объявленной переменной.  ***Пример:***  **public** **class** Solution{  **public** **static** **void** main(String[] args) {  Human human1 = **new** Human();  human1.setName("Volodya");  human1.print();  }  }  **class** Human{  String name;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name){  **this**.name = name;  }  } | Вызов одного конструктора из другого может пригодиться тогда, когда есть несколько конструкторов и не хочется в новом конструкторе переписывать код инициализации, приведенный в конструкторе ранее.  ***Пример***  **class** Human{  **int** age;  **int** weight;  **int** height;  Human(**int** age, **int** weight){  **this**.age = age;  **this**.weight = weight;  }  Human(**int** age, **int** weight, **int** height){  // вы вызываете конструктор с двумя параметрами  **this**(age, weight);  // и добавляете недостающую переменную  **this**.height = height;  }  } |

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

Answer

***final*** - может применяться к классам, методам, переменным (в том числе аргументам методов).

Следует также отметить, что к абстрактным классам (с ключевым словом abstract), нельзя применить модификатор final, т.к. это взаимоисключающие понятия. Для метода final означает, что он не может быть переопределен в подклассах. Это полезно, когда мы хотим, чтобы исходную реализацию нельзя было переопределить.

***Пример:***

**public** **class** SuperClass{

**public** **final** **void** printReport(){

System.out.println("Report");

}

}

**class** SubClass **extends** SuperClass{

**public** **void** printReport(){ // Ошибка компиляции

System.out.println("MyReport");

}

}

Для **переменных примитивного типа** это означает, что однажды присвоенное значение не может быть изменено.

Для **ссылочных переменных** это означает, что после присвоения объекта, нельзя изменить ссылку на данный объект.

**Это важно!** Ссылку изменить нельзя, но состояние объекта изменять можно.

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?

Answer

***Статические поля*** – это поля с ключевым словом **static**. Это означает что эти переменные не экземпляра класса, а принадлежат классу.

***Статические финальные поля*** – это поля с ключевыми словами **static** и **final**. Это означает что эти поля статические (то есть они принадлежат классу) и финальные (то есть изменить переменную после инициализации **нельзя**).

***Статические методы*** – это методы, в шапке которых стоит ключевое слово **static.** Это означает что этот метод принадлежит не экземпляру класса, а именно классу и его нужно вызывать, обращаясь к имени класса.

Статические методы переопределяются и перегружаются.

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?

Answer

***Логические блоки инициализации*** – это блок, использующийся для инициализации обычных переменных.

Пример:

{

//...

}

***Статические блоки инициализации*** – это блок, использующийся для инициализации статических переменных.

Пример:

**static** {

//...

}

Блоков может быть сколько угодно и в каком угодно порядке.

Порядок вывода блоков будет: static block, block, Constructor.

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?

Answer

***Методы с переменным числом параметров*** – это методы у которых во входных параметрах стоит массив с переменной (не фиксированной) длинной и обозначается он у метода следующим образом: type … name.

… - это означает, что длинна не фиксированная.

**Пример:**

class VarArgs {

    static void vaTest(int ... v) { // Метод vaTest() допускает переменное число аргументов

        System.out.println("Число аргументов: " + v.length);

        System.out.println("Содержимое: ");

        for (int i = 0; i < v.length; i++) {

            System.out.println(" arg " + i + ": " + v[i]);

        }

        System.out.println();

    }

    public static void main(String args[]) {

        // Метод vaTest() может вызываться с

        // переменным числом аргументов

        vaTest(1);        // 1 аргумент

        vaTest(1, 2, 3);  // 3 аргумента

        vaTest();         // без аргументов

    }

}

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

Answer

Object – базовый класс для всех классов.

* 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.

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

Answer

Хэш-значение – это значение которое возвращает метод hashCode().

Для методов equals() и hashCode() есть официальные требования, прописанные в документации Oracle:

Если два объекта равны (т.е. метод equals() возвращает true), у них должен быть одинаковый хэш-код.

Иначе наши методы будут лишены смысла. Проверка по hashCode(), как мы и сказали, должна идти первой для повышения быстродействия. Если хэш-коды будут разными, проверка вернет false, хотя объекты на самом деле равны (согласно нашему определению в методе equals()).

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

Answer

***Класс Class*** – это класс для взаимодействия с классами по средствам которого можно получить информацию о классе.

Оператор ***instanceof*** нужен для того, чтобы проверить, был ли объект, на которую ссылается переменная X, создан на основе какого-либо класса Y.

Пример 1:

Main main = **new** Main();

**boolean** b = main **instanceof** Object;

Метод getClass() и Type.class в возвращают класс в котором создан объект.

Пример 2

Class clas = main.getClass();

Таким образом, если сравнивать методом ***instanceof,*** метод возвращает true, если класс объекта, на которую ссылается переменная X (в примере 1 – это main) - это наследник класса Y (в примере 1 – это Object)) (Пример 1). А если сравнивать методами getClass() или Type.class вернётся значение true, если оба метода возвращают один и тот же метод.

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

Answer

***Правила переопределения метода equals():***

Переопределяя метод equals(), обязательно соблюдай эти требования:

1. Рефлексивность.

Любой объект должен быть equals() самому себе. Мы уже учли это требование. В нашем методе указано:

**if** (**this** == object) {

**return** **true**;

}

1. Симметричность.

Если a.equals(b) == true, то и b.equals(a) должно возвращать true.

1. Транзитивность.

Если два объекта равны какому-то третьему объекту, значит, они должны быть равны друг и другу.  
Если a.equals(b) == true и a.equals(c) == true, значит проверка b.equals(c) тоже должна возвращать true.

1. Постоянность.

Результаты работы equals() должны меняться только при изменении входящих в него полей. Если данные двух объектов не менялись, результаты проверки на equals() должны быть всегда одинаковыми.

1. Неравенство с null.

Для любого объекта проверка a.equals(null) должна возвращать false

Это не просто набор каких-то «полезных рекомендаций», а именно жесткий [контракт методов](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Object.html), прописанный в документации Oracle.

Пример переопределения метода equals():

**public** **class** Flower {

**private** String groupName;

**private** String name;

**private** **double** price;

@Override

**public** **boolean** equals(Object object) {

**if** (**this** == object) { // сравниваются облости памяти с объектами

**return** **true**;

}

**if** (object == **null**) { // проверяестся чтобы передаваемый объект не указывал на

**return** **false**; // null

}

**if** (!getClass().equals(object.getClass())) { // сравниваются классы объектов

**return** **false**;

}

Flower flower = (Flower) object; // Понижаю объект класса Object до класса в котором переопределяется метод (в данном случае Flower)

Далее сравниваются поля объекта

Следующим образом сравниваются объекты у полей класса

**if** (groupName == **null**) { // Проверяется чтобы переменная сравниваемого объекта не указывала на null, так как null сравнивать нельзя через equals(), а то будет ошибка NullPointerException.

**if** (flower.groupName != **null**) { // Проверяется чтобы переменная переданного объекта была не равна null

**return** **false**; // В итоге если у одного объекта переменная указывает на null, а у другого не указывает на null, следовательно эти поля не равны и возвращается false

}

} **else** **if** (!groupName.equals(flower.groupName)) { // В ином случае эти поля можно сравнить через equals()

**return** **false**;

}

**if** (name == **null**) {

**if** (flower.name != **null**) {

**return** **false**;

}

} **else** **if** (!name.equals(flower.name)) {

**return** **false**;

}

**if** (price != flower.price) { // Если переменная поля примитивного типа, то нужно сравнимать сссылки

**return** **false**;

}

**return** **true**;

}

}

***Правила переопределения метода hashCode():***

Как для equals() так и для метода hashCode() есть официальные требования, прописанные в документации Oracle:

1. Если два объекта равны (т.е. метод equals() возвращает true), у них должен быть одинаковый хэш-код.

Иначе наши методы будут лишены смысла. Проверка по hashCode(), как мы и сказали, должна идти первой для повышения быстродействия. Если хэш-коды будут разными, проверка вернет false, хотя объекты на самом деле равны.

1. Если метод hashCode() вызывается несколько раз на одном и том же объекте, каждый раз он должен возвращать одно и то же число.
2. Правило 1 не работает в обратную сторону. Одинаковый хэш-код может быть у двух разных объектов.

***Пример переопределения метода hashCode():***

**public** **class** Flower {

**private** String groupName;

**private** String name;

**private** **double** price;

**private** **static** **final** String ***LINE\_GROUP\_FLOWER*** = "Group of flower: ";

**private** **static** **final** String ***LINE\_NAME*** = "Name: ";

**private** **static** **final** String ***LINE\_PRICE*** = "Price: ";

**private** **static** **final** String ***OPERATION\_NEXT\_LINE*** = "\n";

@Override

**public** **int** hashCode() {

**final** **int** prime = 1;

**int** result = 31;

result = result \* prime + (groupName != **null** ? groupName.hashCode() : 1);

result = result \* prime + (name != **null** ? name.hashCode() : 1);

result = result \* prime + Double.*hashCode*(price);

result = result \* prime + (***LINE\_NAME*** != **null** ? ***LINE\_NAME***.hashCode() : 1);

result = result \* prime + (***LINE\_PRICE*** != **null** ? ***LINE\_PRICE***.hashCode() : 1);

result = result \* prime + (***LINE\_GROUP\_FLOWER*** != **null** ? ***LINE\_GROUP\_FLOWER***.hashCode() : 1);

result = result \* prime + (***OPERATION\_NEXT\_LINE*** != **null** ? ***OPERATION\_NEXT\_LINE***.hashCode() : 1);

**return** result;

}

}

***Переопределение метода toString():***

Метод *toString(),* для того чтобы когда вызывается метод *toString(),* то есть вывод информации о объекте, выводимая информация должна быть информативная, понятная.

Данный метод переопределён в классе Object. Мы переопределяем метод *toString(),* чтобы информация выводилась про конкретный объект.