Объектная ориентация Java

1. **Все есть объект.** Думать об объектах, как об особенных переменных; они хранят данные и можно “сделать запрос” к такому объекту, попросив его самого выполнить операцию.
2. **Программа - это связка объектов, говорящих друг другу что делать, посылая сообщения.** Чтобы сделать запрос к объекту, надо “посылать сообщение” этому объекту. Правильнее думать о сообщении, как о запросе на вызов функции, которая принадлежит определенному объекту.
3. **Каждый объект имеет свою собственную память, отличную от других объектов.**
4. **Каждый объект имеет тип.** Другими словами, каждый объект является *экземпляром класса*, где “класс” - это синоним “типа”.
5. **Все объекты определенного типа могут принимать одинаковые сообщения.**

Описание класса

[модификаторы] class ИмяНовогоКласса [extends ИмяСупер-КлассаБазового класса] [implements список Интерфейсов]

{Тело класса, состоящее из описаний элементов класса}

Модификаторы доступа - public, protected, private, package-private **(**default по умолчанию - уровень пакета)

Модификаторы использования - final, abstract, static

Доступность данных и методов класса в зависимости от места их размещения

Откуда возможен доступ к элементам / Спецификатор доступа	Данный класс	Другие классы в пакете, которому принадлежит класс	Классы в других пакетах	
			Потомки	Не потомки
public	+	+	+	+
protected	+	+	+	-
private	+	-	-	-
по умолчанию	+	+	-	-

1. Компоненты класса, объявленные как private, доступны только в своем классе.
2. Компоненты класса доступны из другого класса того же пакета, если они не объявлены как private.
3. Компонент класса A доступен из класса B, входящего в другой пакет, в тех случаях, когда класс A, равно как и его компонент, объявлены как public, или когда класс A объявлен как public, компонент класса объявлен как protected, а класс B является подклассом класса A.

Модификаторы использования класса

- final - класс не может быть расширен (не может быть базовым)
- abstract - класс не предназначен для создания объектов. Абстрактный класс может использоваться только как базовый для классов-наследников. Класс, содержащий хотя бы один абстрактный метод, должен быть объявлен абстрактным. Модификаторы final и abstract несовместимы.
- static – используется для вложенных классов, в которых имеются статические переменные и методы.

Перечисление – это класс

enum Season {WINTER, SRRING, SUMMER, AUTUMN}

Эквивалентно:

class Season extends java.lang.Enum

{WINTER, SRRING, SUMMER, AUTUMN}

Экземпляры enum-класса имеют права по умолчанию, доступны статически и наследуют методы:

— name() — имя константы в виде строки

— ordinal() — порядок константы (соответствует порядку, в котором объвлены константы)

— values() — возвращает массив всех значений перечисления;

Season season = Season. WINTER;

System.out.println(“Вывод =”+season.name());

Вывод= WINTER

Переменные класса

Описание переменной выполняется по следующей схеме:

[модификаторы] Тип ИмяПеременной [=значение];

Модификаторами могут быть :

- любой из модификаторов доступа: public,protected,private, иначе уровень доступа переменной устанавливается по умолчанию пакетным.
- static – модификатор принадлежности – переменные, отмеченные этим модификатором, принадлежат классу, а не экземпляру класса и существуют в единственном числе для всех его объектов, создаются JVM в момент первого обращения к классу, допускают обращение до создание объекта класса

  Обращение: **Имя класса.Имя компонента**
- final – переменная не может изменять своего начального значения, то есть, является именованной константой.
- transient – переменная не должна сохранять и восстанавливать значение при сериализации (записи в файл) объекта. Все статические переменные являются несохраняемыми автоматически.
- volatile – Используется в многопоточном программировании. Сообщает компилятору, что к полю могут одновременно обращаться несколько потоков текущего процесса. Запрещает оптимизирующему компилятору использовать ее копии, размещаемые в регистрах и кэшах процессоров.

Инициализация полей в Java

- Инициализация статических полей и блоков выполняется при загрузке класса
- Инициализация не статических элементов выполняется при создании объекта (при вызове конструктора)

Порядок инициализации полей

- инициализация статических полей в месте объявления и в инициализационном блоке происходит до инициализации в конструкторе
- инициализации полей в месте объявления и в инициализационных блоках выполняются в порядке их объявления в классе
- инициализация полей базового класса происходит полностью до инициализации производного класса, т.е. сначала выполняются все инициализаторы базового класса, а потом все инициализаторы производного класса.

Пример инициализации полей

```
public class ClassFieldsInitOrder {
int i0=1;
static int i1 = initialize(“i1");
static int i2;
static { i2 = initialize("i2"); }
static int i3 = initialize(“i3");
static int i4;
static { i4 = initialize("i4"); }
static int initialize(String name) { System.out.println(name); return 0; }
public static void main(String[] args) {
 System.out.println(“i0=“+i0+“ i1=”+i1+” i2=“+i2+” i3=”+i3+”i4=”+i4);
}
}
```

i1 i2 i3 i4 // при загрузке класса

i0=1 i1=0 i2=0 i3=0 i4=0 // при выполнении метода main

Методы класса

[модификаторы] ТипВозвращаемогоЗначения ИмяМетода (список Параметров) [throws списокВыбрасываемыхИсключений] { Тело метода};

Модификаторами могут быть

- любой из модификаторов доступа: public,protected,private, иначе уровень доступа метода устанавливается по умолчанию пакетным.
- static – модификатор принадлежности – методы, отмеченные этим модификатором, принадлежат классу и доступны до создания объектов. Статические методы могут оперировать только статическими переменными или локальными переменными, объявленными внутри метода. Статический метод не может быть переопределен.

class A {

. . .

public static void main(String[] args) { . . . } – программа загружается как класс java A, а не как объект. Затем вызывается метод A.main (args)

- final – метод не может быть переопределен при наследовании класса.
- synchronized – при исполнении метода не может произойти переключение конкурирующих потоков
- abstract – нереализованный метод. Тело такого метода просто отсутствует. Если объявили некий метод класса абстрактным, то и весь класс надо объявить абстрактным.

Назначение конструкторов

Конструктор – это метод, назначение которого состоит в создании экземпляра класса.

Характеристики конструктора:

- Имя конструктора должно совпадать с именем класса;
- Если в классе не описан конструктор, компилятор автоматически добавляет в код конструктор по умолчанию;
- Конструктор не может быть вызван иначе как оператором new;
- Конструктор не имеет возвращаемого значения – так как он возвращает ссылку на создаваемый объект (допускается оператор return, но только пустой).
- Конструкторов может быть несколько в классе. В этом случае конструкторы называют перегруженными;

Отличие конструкторов от методов

Свойство	Конструкторы	Методы
Назначение	Создает экземпляр класса	Группирует операторы Java
Модификаторы	Не может быть abstract, final, static, или synchronized	Может быть abstract, final, static, или synchronized
Возвращаемый тип	Нет возвращаемого типа, не может быть даже void	void или любой корректный тип
Имя	Такое же как и имя класса (по договоренности, первая буква — заглавная) — обычно существительное	Любое имя, за исключением имени класса. Имена методов начинаются со строчной буквы, по договоренности, и обычно являются глаголами
this	Ссылается на другой конструктор в этом же классе. Если используется, то обращение должно к нему быть первой строкой конструктора	Ссылается на экземпляр класса-владельца. Не может использоваться статическими методами
super	Вызывает конструктор родительского класса. Если используется, должно обращение к нему быть первой строкой конструктора	Вызывает какой-либо переопределенный метод в родительском классе
наследование	Конструкторы не наследуются	Методы наследуются
Автоматическое добавление кода конструктора компилятором	Если в классе не описан конструктор, компилятор автоматически добавляет в код конструктор без параметров	Отсутствует
Автоматическое добавление компилятором вызова конструктора класса-предка	Если конструктор не делает вызов конструктора super класса-предка (с аргументами или без аргументов), компилятор автоматически добавляет код вызова конструктора класса-предка без аргументов	Отсутствует

Правила доступа к конструкторам

Модификатор доступа к конструктору	Описание
`public`	Конструктор может быть вызван любым классом.
`protected`	Конструктор может быть вызван классом из того же пакета или любым подклассом.
Без модификатора	Конструктор может быть вызван любым классом из того же пакета.
`private`	Конструктор может быть вызван только тем классом, в котором он определен.

Последовательность действий при вызове конструктора

1. Все поля данных инициализируются своими значениями, предусмотренными по умолчанию (0, false или null).
2. Инициализаторы всех полей и блоки инициализации выполняются в порядке их перечисления в объявлении класса.
3. Если в первой строке конструктора вызывается другой конструктор, то выполняется вызванный конструктор.
4. Выполняется тело конструктора.

Конструкторы по умолчанию

```
public class Konstr {
int width; // ширина коробки
    int height; // высота коробки
    int depth; // глубина коробки
    // вычисляем объём коробки
    int getVolume() {
    return width * height * depth;
    }
public static void main(String[] args) {
Konstr kons=new Konstr(); // вызов конструктора по умолчанию
System.out.println("Объём коробки: " + kons.getVolume());
}
}
```

Всем переменным присваивается значения по умолчанию (0)

Объём коробки: 0

Конструктор без параметров

```
public class Konstr {
int width; // ширина коробки
   int height; // высота коробки
   int depth; // глубина коробки
Konstr() {    // конструктор без параметров
      width = 10;
      height = 10;
      depth = 10;
   }
   // вычисляем объём коробки
   int getVolume() {
   return width * height * depth;
   }
public static void main(String[] args) {
Konstr kons=new Konstr(); // вызов конструктора без параметров
System.out.println("Объём коробки: " + kons.getVolume());
}
}
```

Объём коробки: 1000

Конструктор с параметрами

```
public class Konstr {
int width; // ширина коробки
   int height; // высота коробки
   int depth; // глубина коробки
      Konstr(int w, int h, int d) {// конструктор с параметрами
      width = w;
   height = h;
   depth = d;
   }
 // вычисляем объём коробки
int getVolume() {
   return width * height * depth;
   }
public static void main(String[] args) {
Konstr kons=new Konstr(5,5,5); // вызов конструктора с параметрами
System.out.println("Объём коробки: " + kons.getVolume());
Konstr kons1=new Konst(); // ошибка, конструктор не определен
}
}
```

Объём коробки: 125

Перегрузка конструкторов

```
 Konstr() {    // конструктор без параметров
       width = 10;
       height = 10;
       depth = 10;
   }
Konstr(int w, int h, int d) {   // конструктор с параметрами
       width = w;
   height = h;
   depth = d;
   }
public static void main(String[] args) {
Konstr kons1=new Konstr(); // вызов конструктора без параметров
System.out.println("Объём стандартной коробки: " + kons1.getVolume());
Konstr kons2=new Konstr(5,5,5); // вызов конструктора с параметрами
System.out.println("Объём нестандартной коробки: " + kons2.getVolume());
}
```

Объём стандартной коробки: 1000
Объём нестандартной коробки: 125

Конструктор копирования

```
Konstr(int w, int h, int d) {   // конструктор с параметрами
       width = w;
    height = h;
    depth = d;
    }
Konstr(Konstr ob) {  // конструктор копирования
width = ob.width;
height = ob.height;
depth = ob.depth;
}
public static void main(String[] args) {
Konstr kons2=new Konstr(7,7,7); // вызов конструктора с
    параметрами
Konstr kons1=new Konstr(kons2); // вызов конструктора
    копирования
System.out.println("Объём новой коробки: " + kons1.getVolume());
}
```

Объём новой коробки: 343

Вызов перегруженных конструкторов через this()

```
Konstr(int w, int h, int d) {   // конструктор с 3 параметрами
       width = w;
   height = h;
   depth = d;
   }
Konstr(int i) {   // конструктор с 1 параметром
/* Вызов конструктора this() должен быть первым оператором в
     конструкторе.*/
     this (i,i,i); вызов конструктора с 3 параметрами
   width++;
   height--;
   depth ++;
   }
public static void main(String[] args) {
Konstr kons=new Konstr(3); // вызов конструктора с 1 параметром
System.out.println("Объём коробки: " + kons.getVolume());
}
```

Объём коробки: 32

Конструкторы с переменным числом параметров

```
Konstr(int ... i) {   // конструктор с переменным числом параметров
   int j=0;
   for (int v : i) {
   switch (j) {
   case 0: width=v; j++; break;
   case 1: height=v; j++; break;
   case 2: depth=v; j++; break;
   default: break;
   }
   }
    if (j==0){
      width = 1;
      height = 1;
      depth = 1;
   }
   if (j==1){
       height = 1;
      depth = 1;
   }
   if (j==2){
   depth = 1;
   }
   }
Konstr kons0=new Konstr();          // вызов конструктора без параметров
Konstr kon1s=new Konstr(2);         // вызов конструктора с 1 параметром
Konstr kons2=new Konstr(4,3);       // вызов конструктора с 2 параметрами
Konstr kons3=new Konstr(1,2,3);     // вызов конструктора с 3 параметрами
```