Объектная ориентация Java

1. **Все есть объект.** Объект — это мыслимая или реальная сущность, обладающая характерным поведением и отличительными характеристиками и являющаяся важной в предметной области.
2. **Программа - это связка объектов, говорящих друг другу что делать, посылая сообщения.** Чтобы сделать запрос к объекту, надо послать сообщение этому объекту (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.**

Свойства ООП

- **Абстракция**

Выделение важных сущностей и свойств реального мира для их моделирования в программе

- **Инкапсуляция**

Объединения внутри класса данных и операций с ними, их защита от прямого доступа

- **Наследование**

Возможность создания производных классов на основе существующих (базовых)

- **Полиморфизм**

Возможность использования производного класса, там где разрешен экземпляр базового класса, при этом вызываются методы, переопределенные в производном классе

Описание класса

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

{Тело класса, состоящее из описаний элементов класса}

Модификаторы доступа – **public** (доступен из любого другого класса, расположенного в каких-либо других пакетах), **default** (по умолчанию, доступен только из других классов в одном пакете), **protected** (видимость в границах пакета и видимость для классов-наследников), **private** (обеспечивает видимость только в границах класса).

Для объявления класса верхнего уровня могут быть использованы только модификатор доступа **public** или **default**. Можем иметь только один «public» класс в исходном файле. Имя файла должно быть таким же, как имя public-класса.

Модификаторы использования - **final, abstract, static**

Модификаторы использования класса

- **final** - класс не может быть расширен (не может быть базовым для производного)
- **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.

Создание объектов

Для создания объекта необходимо объявить переменную типа класс и создать экземпляр (объект) класса с помощью оператора new.

ИмяКласса имяОбъекта; // объявление объекта

// создание объекта

имяОбъекта=new ИмяКласса ([параметры конструктора]);

class Student { ……} // описание класса

Student ivanov; // объявление объекта Иванов

ivanov=new Student(); // создания объекта Иванов

Student petrov=new Student(); // объявление и создание объекта Петров

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. 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.

Правила объявления переменных

- Имена переменных не могут начинаться с цифры, в именах не могут использоваться как символы арифметических и логических операторов, а также символ ‘#’.
- Применение символов ‘$’ или ‘_’ приемлемо, включая первую позицию.
- Переменная, объявленная как член класса и глобальная (static) переменная, по умолчанию задается нулем.
- Если переменная объявлена как локальная переменная в методе, перед использованием она должна обязательно быть проинициализирована. Так как локальные переменные не инициализируются по умолчанию. в то время, как создав такую переменную как член класса (глобальную) компилятор сам задаст ей значение 0.
- Область действия и время жизни переменной ограничено блоком {}, в котором она объявлена. Глобальную переменную видно во всех блоках.

Пример описания переменных класса

```
public class Student {
private String name;
 private  int group;
 public String specialty;
public static String univer;
}
class Student petrov=new Student();
petrov.specialty="Информатика и ВТ";
Student.univer="ЛЭТИ";
String c=petrov.univer;
```

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 класса или своими локальными переменными. Не доступен this и 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 нельзя вызвать этот метод из другого потока для данного объекта
- abstract – нереализованный метод. Тело такого метода отсутствует. Если объявили некий метод класса абстрактным, то и весь класс надо объявить абстрактным.

Сравнение обычных и статических методов

Способность	Обычный метод	Статический метод
Есть связь с экземпляром класса	Да	Нет
Может вызывать обычные методы класса	Да	Нет
Может вызывать статические методы класса	Да	Да
Может обращаться к обычным переменным класса	Да	Нет
Может обращаться к статическим переменным класса	Да	Да
Может быть вызван у объекта	Да	Да
Может быть вызван у класса	Нет	Да

Пример описания методов класса

```
public class Student {
    private String FIO;
    private int group;
    public String getFIO() {
        return FIO;
    }
    public void setFIO(String _FIO) {
        FIO = _FIO;
    }
    public int getGroup() {
        return group;
    }
    public void setGroup(int _group) {
        group = _group;
    }
    static void metod() {    //вывод информации о студенте в консоль
        System.out.println(FIO + " " + group);
    }
public static void main(String[] args) {
metod(); //Student.metod();
}
}
```

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.
4. Правила доступа для членов класса будут применимы только после правил доступа на уровне класса.

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 конструктора	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 объекта 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. Дефолтный
     конструктор исчезает из класса, когда создается какой-то конструктор с
     аргументами.
}
}
```

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

В отличие от C++, Java не предоставляет прямые конструкторы 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, i массив целых чисел
   int j=0;
   for (int v : i) { // переменной v автоматически присваивается значение, равное значению следующего элемента
                     массива 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
```