Мультимедийный курс

Программирование на Java

Лекция 3.2

Объектная модель JAVA (ч. 2)

1. ***Пакеты***
2. ***Класс Object***
3. ***Наследование***

Пакет - некоторый набор родственных (связанных) классов (аналог подкаталога жесткого диска)

- служит для управления видимостью
- позволяет разделить пространство имен

Создание пакета

Вначале файла исходного текста (можно после комментариев) включается оператор

package <*имя_пакета*>;

- при отсутствии такого оператора классы файла относят к **пакету без имени**
- на один и тот же пакет могут ссылаться разные файлы. Все они должны находиться **в одном каталоге**

Иерархия пакетов

Задается конструкцией

package *<имя_пак.1>* [. … [. *<имя_пак.N>*]];

package proj.mod3.sect1;

Пакет, входящий в иерархию, должен храниться в соответствующем подкаталоге файловой системы

<имя_пак.1> / … / *<имя_пак.N>*

Использование классов пакета

Варианты:

а) *при обращении к классу указать реальное имя класса с учетом имени пакета*

<имя_пакета>.<имя_класса>

```
class  MyDate  extends  java.util.Date;
```

Использование классов пакета

б) *использовать оператор импорта пакета*

```
import <имя_пакета>.<имя_класса>;
```

или

```
import <имя_пакета>.*;
```

Примеры:

import java.util.Date;

import java.io.;*

Все классы Java (даже массивы) происходят от **суперкласса** java.lang.Object

Методы класса Object

- методы поддержки многопоточности
- прикладные методы

Прикладные методы

Наименование метода	Описание
Boolean equals(Object object)	*Проверяет идентичность ссылок текущего и заданного объектов*
int hachCode()	*Возвращает значение хеш-кода*
Class getClass()	*Определяет тип объекта. Возвращает объект-описатель класса*
String toString()	*Возвращает строку из имени класса и адреса объекта*
Object clone()	*Клонирует объект посредством побитного копирования полей*
void finalize()	*Деинициализует объект. Вызывается перед уничтожением объекта сборщиком мусора*

Метод boolean equals(Object obj)

Метод equals() следует переопределять тогда, когда для класса существует понятие логической эквивалентности, которое не совпадает с тождественностью объектов

Метод equals должен удовлетворять следующим требованиям:

- *Рефлексивность*. **x.equals(x)** всегда должно быть true.
- *Симметричность*. Если **x.equals(y) == true**, то и **y.equals(x)** должно быть **true**
- *Транзитивность*. Если **x.equals(y) == true**, **y.equals(z) == true** то и **x.equals(z)** должно быть **true**
- *Непротиворечивость*. **x.equals(null)** всегда должно возвращать **false**

Метод boolean equals(Object obj). Пример

```
public class Point {
    private final int x;
    private final int y;
    public Point(int x, int y){
        this.x = x;
        this.y = y;
    }

    public boolean equals(Object obj){
      if (obj == this){
          return true;
      }
       if (!(obj instanceof Point)){
           return false;
       }
       Point p = (Point)obj;
       return p.x == x && p.y == y;
    }
}
```

equals() vs. ==

```
String a="abc";
String b="abc";
String c=new
  StringBuffer("a").append("bc").toString();
```

a == b ?

a == c ?

a.equals(b) ?

a.equals(c) ?

Метод hashCode()

Метод **int hashCode()** применяется при построении хеш-таблиц (HashSet, HashMap, и др.)

Требования к методу:

- Он должен быть переопределен в каждом классе, где переопределен метод **equals()**
- При нескольких обращениях к объекту метод должен возвращать одно и то же число
- Если два объекта равны по результатам метода **equals()**, их хеш-коды тоже должны быть равны
- Если два объекта не равны по **equals()**, они не обязательно должны иметь разный хеш-код. Но это очень желательно

```
public int hashCode(){
  int result = 17;
  result = 37*result + areaCode;
  result = 37*result + extension;
  return result;
}
```

Метод toString()

Метод **toString()** возвращает строковое представление объекта

Желательно, чтобы все классы переопределяли этот метод

В классе Object он формирует строку как

getClass().getName() + '@' + Integer.toHexString(hashCode())

Этот метод вызывается при конкатенации строк, если указывается ссылка на объект, а также при печати объекта на консоль

```
public class Point {
      private int x, y;
      Point(int x, int y){
         this.x= x;
         this.y= y;
     }
     public String toString(){
         return x+", "+ y + ")";
    }
    public static void main(String[] args){
        System.out.println(new Point(1, 3)); // Распечатает (1, 3)
        String str = "Point " + new Point(2, 3);
        System.out.println(str); // Распечатает Point (2, 3)
     }
}
```

Вывод информации об объекте

System.out.Println("Объект "+x);

x.toString();

Метод finalize()

Метод **finalize()** вызывается сборщиком мусора непосредственно перед удалением объекта

Как правило, используется для освобождения ресурсов, используемых объектом

Он не должен содержать никаких критических по времени операций, поскольку **время удаления объекта не регламентировано**

На практике вместо того, чтобы помещать всю логику по освобождению ресурсов в этот метод, используйте метод прямого завершения (пример - метод **close()** у **java.io.OutputStream**)

Метод **finalize()** следует применять только **в качестве запасного варианта** для освобождения ресурсов (проверять, был ли вызван метод прямого завершения, и если нет – вызывать его)

Клонирование объектов

Метод clone - protected

```
Employee orig= new Employee("Иван Иванов", 4500);
Employee copy = orig.clone();
```

Ссылка на подобъект едина для оригинала и клонированного объекта **!!!**

Наследование – использование одним классом структуры или поведения другого *класса* (одиночное наследование)

Наследование способствует

- **созданию иерархических классификаций**
- **уменьшению количества кода, созданного для описания схожих сущностей,**
- **написанию более эффективного и гибкого кода**

Суперкласс – **унаследованный класс**

Подкласс – **специализированная версия суперкласса, которая, обычно, дополняет или переопределяют унаследованную структуру и поведение**

Объявление наследования:

в определение подкласса включается

extends <*имя_суперкласса*>

Пример:

```
class Rect extends Figure{
 …
}
```

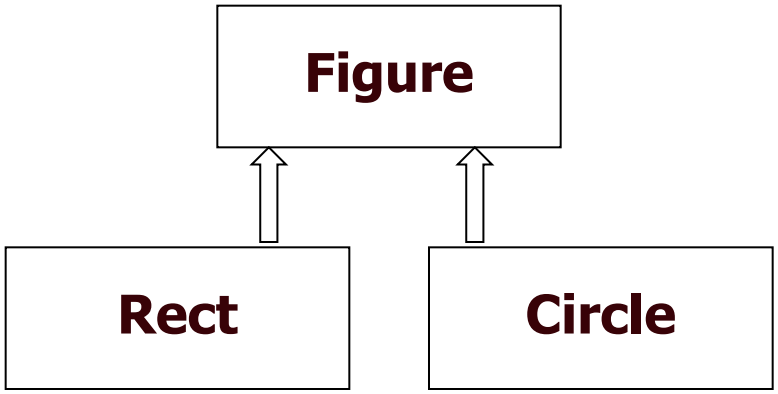
Пример.

```
// Суперкласс
class Figure {
    int x, y;            //  координаты абстрактной фигуры
    public Figure(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public void move (int dx, int dy) {    // передвижение фиг.
        x += dx;
        y += dy;
    }
    public void print() {
        System.out.print("Figure: " + x + "  " + y);
    }
}
```

```
// Подкласс - прямоугольник
class Rect extends Figure{
    int w, h;       // ширина и высота
    public Rect(int x1, int y1, int x2, int y2) {
        super(x1, y1);
        this.w = x2 - x1;
        this.h = y2 - y1;
    }
    public void print() {
      System.out.print("Rectangle: " + x + "  " + y + "  "
+ w + "  " + h);
    }
}
```

```
//   Подкласс - окружность
class Circle extends Figure{
    int r;          // радиус
    public Circle(int x, int y, int r) {
        super(x, y);
        this.r = r;
    }
    public void print() {
        System.out.print("   Circle: " + x + "  " + y +
                                                "  " + r);
    }
}
```

Созданная структура классов




```
...
Rect rect = new Rect(10,10,100,200);
rect.print();
rect.move(20,10);
rect.print();
…
```

Результат

```
Rectangle: 10  10  90  190
Rectangle: 30  20  90  190
```

В производном классе можно объявлять *поля, одноименные с полями базового класса*

Тогда

- объект производного класса будет содержать по два поля
- производное имя будет скрывать базовое

Обращение из метода производного класса к полю суперкласса:

Методы, одноименные с методами базового класса

- при совпадении параметров производный метод перекроет базовый
- при несовпадении параметров производный метод перегрузит базовый

Перекрытый метод - недоступен в подклассе

Перегруженный метод - виден наряду с новым методом. Вызов из метода подкласса:

super.<*имя_метода*>()

Последовательность вызова конструкторов

Вызов (при необходимости) конструктора суперкласса

super (<*список_параметров*>)

Пример. Иллюстрация последовательности действий при конструировании объекта:

```
…
B b1=new B();

…
class A {
    { System.out.print(1); }
    A(){  System.out.print(2);  }
}
class B extends A {
    { System.out.print(3); }
    B(){ System.out.print(4);  }
}
```

Результат: "1234"

Преобразование между типами, связанными отношением наследования

1) **восходящее**: от подкласса к суперклассу - выполняется неявно

2) **нисходящее**: от суперкласса к подклассу - выполняется при помощи операции приведения типа:

```
D d= (D) z;
```

Проверка допустимости преобразования:

оператор **instanceof**

```
// B - базовый класс  D - производный класс
B b = new D();
D d;
 if (b instanceof D)
          d = (D) b;
```