Структури от Данни и Обектно-Ориентирано Програмиране
спец. Компютърни Науки
Упражнение №1
25.02.2010г.

ТЕМА: Преговор. Наследяване и полиморфизъм

Какво може да се прави в наследника на даден клас

- Да се използва наследено поле;
- Да се декларира поле с името на поле от базовия клас, при което последното се ***скрива***;
- Да се декларират нови полета;
- Да се ползват наследените методи;
- Да се създава нов метод на обекта, който има сигнатурата на метод от базовия клас, при което последният се ***припокрива***. При това типовете на връщаните стойности трябва да съвпадат;
- Да се създава нов метод на класа, който има сигнатурата на метод от базовия клас, при което последния се ***скрива***;
- Да се декларират нови методи;

Ред на изпълнение на конструкторите при наследяване

Пример 1:

```
class Meal {
  Meal() { System.out.println("Meal()"); }
}

class Bread {
  Bread() { System.out.println("Bread()"); }
}

class Cheese {
  Cheese() { System.out.println("Cheese()"); }
}

class Lettuce {
  Lettuce() { System.out.println("Lettuce()"); }
}
```

```
class Lunch extends Meal {
  Lunch() { System.out.println("Lunch()");}
}

class PortableLunch extends Lunch {
  PortableLunch() {
    System.out.println("PortableLunch()");
  }
}

class Sandwich extends PortableLunch {
  Bread b = new Bread();
  Cheese c = new Cheese();
  Lettuce l = new Lettuce();
  Sandwich() {
    System.out.println("Sandwich()");
  }

  public static void main(String[] args) {
    new Sandwich();
  }
}
```

```
Meal()
Lunch()
PortableLunch()
Bread()
Cheese()
Lettuce()
Sandwich()
```

Пример 2:

```
class Base {
        int field;

        Base() {
                field = 0;
                System.out.println("Creating base object, field = " + field);
                method();
        }
```

```
        void method() {
                System.out.println("Here I am in base method!");
        }
}

class Derived extends Base {
        int field;

        Derived() {
                field = 1;
                System.out.println("Creating derived object, field = " + field);
                method();
        }

        void method() {
                System.out.println("Here I am in derived method!");
        }
}

public class ObjectCreation {

        public static void main(String[] args) {
                Base b = new Base();
                System.out.println();
                Derived d = new Derived();
        }
}
```

Полиморфно поведение на методи на обекта

```
class Actor {
  public void act() {}
}

class HappyActor extends Actor {
  public void act() { System.out.println("HappyActor"); }
}

class SadActor extends Actor {
  public void act() {  System.out.println("SadActor"); }
}
```

```
class Stage {
  private Actor actor = new HappyActor();
  public void changeRole() { actor = new SadActor(); }
  public void performPlay() { actor.act(); }
}

public class Transformation {
  public static void main(String[] args) {
    Stage stage = new Stage();
    stage.performPlay();
    stage.changeRole();
    stage.performPlay();
  }
}
```

HappyActor
SadActor

Наследяване на полета и методи. Полиморфизъм

Пример 1:

```
class Super {
        public int field = 0;
        public int getField() { return field; }
}

class Sub extends Super {
        public int field = 1;
        public int getField() { return field; }
        public int getSuperField() { return super.field; }
}

public class FieldAccess {
        public static void main(String[] args) {
                Super sup = new Sub(); // upcast
                System.out.println("sup.field = " + sup.field + ", sup.getField() = " +
                                                                sup.getField());
                Sub sub = new Sub();
                System.out.println("sub.field = " + sub.field + ", sub.getField() = " +
                                sub.getField() +", sub.getSuperField() = " + sub.getSuperField());
        }
}
```

Пример 2:

```
public class BaseClass {

        int x;
        int z;

        public BaseClass() {}

        public void method1() {
          System.out.println("Base method1");
        }

        public int x() {
                return x;
        }

        public BaseClass method4() {  // ???
                return new BaseClass();
        }
}

public class Derived extends BaseClass {
        int x=1;
        int y=2;
        float z=3;

        public Derived(){}

        public int x() {return x;}

        public void method1() {
                  System.out.println("Derived method1");
        }


        /* Overrides method with different return type
        public int method1() {
          System.out.println("Derived method1");
          return 1;
        }
        */
```

```
        public void method2() {
          System.out.println("Derived method2");
        }

        public void method3() {
          System.out.println(x);
          System.out.println(super.x);
        }

        public Derived method4() {
                return new Derived();
        }
}

public class Main {

       public static void main(String[] args) {
          BaseClass [] x= { new BaseClass(),  new Derived() };
          x[0].method1();  // Base method1
          x[1].method1();  // Derived Method1
          // x[0].method2();  // Compile Time Error
          // ((Derived)x[0]).method2();  // Class Cast Exception
          // x[1].method2();  // Compile Time Error
          ((Derived)x[1]).method2();
          ((Derived)x[1]).method3();
          System.out.println(x[0].x+" "+x[1].x+" "+((Derived)x[1]).y);  // 0 0 2
          System.out.println(x[0].x+" "+((Derived)x[1]).x);  // 0 1
          System.out.println(x[0].x()+" "+x[1].x()+" "+((Derived)x[1]).y);  // 0 1 2
          System.out.println(x[0].z+" "+x[1].z+" "+((Derived)x[1]).z);  // 0 0 3.0
       }
}
```

Обекти. Претоварване, припокриване и скриване

Пример 1:

```
class Point {
        int x = 0, y = 0;
        void move(int dx, int dy) { x += dx; y += dy; }
        int color;
}
```

```
class RealPoint extends Point {
        float x = 0.0f, y = 0.0f;
        void move(int dx, int dy) { move((float)dx, (float)dy); }
        void move(float dx, float dy) { x += dx; y += dy; }
}
```

Пример 2: Грешка при наследяване

```
class Point {
        int x = 0, y = 0, color;
        void move(int dx, int dy) { x += dx; y += dy; }
        int getX() { return x; }
        int getY() { return y; }
}

class RealPoint extends Point {
        float x = 0.0f, y = 0.0f;
        void move(int dx, int dy) { move((float)dx, (float)dy); }
        void move(float dx, float dy) { x += dx; y += dy; }
        float getX() { return x; }
        float getY() { return y; }
}
```

Наследяване на полета съобразно модификатор за достъп

```
public class BaseClass {

        public int aaa = 1;
        double bbb = 2;
        protected float ccc = 3;
        private long ddd = 4;

        public BaseClass() {}

        public long methodD() {
           return ddd;
        }
}

public class Derived extends BaseClass {
        int x = 1;
        int y = 2;
```

```
        float z = 3;
        public Derived(){}
        public int x() {return x;}
}

public class Main {

      public static void main(String[] args) {
         BaseClass [] x= { new BaseClass(),  new Derived() };
         System.out.println(x[0].aaa+" "+x[1].aaa);  // 1 1
         System.out.println(x[0].bbb+" "+x[1].bbb);  // 2.0 2.0
         System.out.println(x[0].ccc+" "+x[1].ccc);  // 3.0 3.0
         // System.out.println(x[0].ddd+" "+x[1].ddd);  // Field not visible
         System.out.println(x[0].methodD()+" "+x[1].methodD());  // 4 4
      }
}
```

Претоварване, припокриване и скриване на методи при наследяване

Пример 1

```
public class Tree {
        int height;

        Tree() {
          prt("Planting a seedling");
          height = 0;
        }

        Tree(int i) {
          prt("Creating new Tree that is " + i + " feet tall");
          height = i;
        }

        void info() {
          prt("Tree is " + height + " feet tall");
        }

        void info(String s) {
          prt(s + ": Tree is " + height + " feet tall");
        }
```

```
    static void prt(String s) {
      System.out.println(s);
    }
}

public class Overloading {

    public static void main(String[] args) {
        for(int i = 0; i < 5; i++) {
            Tree t = new Tree(i);
            t.info();
            t.info("overloaded method");
        }
        new Tree();
    }
}

public class OverloadingOrder {

    static void print(String s, int i) {
        System.out.println("String: " + s + ", int: " + i);
    }

    static void print(int i, String s) {
        System.out.println("int: " + i + ", String: " + s);
    }

    public static void main(String[] args) {
      print("String first", 11);
      print(99, "Int first");

    }
}
```

Пример 2:

```
public class A {
  String name() { return "A"; }
}

public class B extends A {
    String name() { return "B"; }
}
```

```
public class C extends A {
	String name() { return "C"; }
}

public class D extends C {
	public D() {}
}

public class Overriding {

	public static void main(String[] args) {
		A[] tests = new A[] { new A(), new B(), new C(), new D() };
		for (int i = 0; i < tests.length; i++)
		System.out.print(tests[i].name());
	}
}
```

Пример 3:

```
public class Super {
	static String greeting() { return "Goodnight"; }
	String name() { return "Kuku"; }
}

public class Sub extends Super {
	static String greeting() { return "Hello"; }
	String name() { return "Pipi"; }
}

public class Test {
	public static void main(String[] args) {
		Super s = new Sub();
		System.out.println(s.greeting() + ", " + s.name());
		System.out.println(Super.greeting() + ", " + s.name());
		Super s1 = new Super();
		System.out.println(s1.greeting() + ", " + s1.name());
		System.out.println(Sub.greeting() + ", " + s.name());
	}
}
```

```
Goodnight, Pipi
Goodnight, Pipi
Goodnight, Kuku
Hello, Pipi
```

Пример 4:

```
public class Animal {

   public static void testClassMethod() {
      System.out.println("The Animal class method");
   }

   public void testInstanceMethod() {
      System.out.println("The Animal instance method");
   }

}

public class Cat extends Animal {

   public static void testClassMethod() {
      System.out.println("The Cat class method");
   }
   public void testInstanceMethod() {
      System.out.println("The instance Cat method");
   }

   public static void main(String[] args) {
      Cat myCat = new Cat();
      Animal myAnimal = myCat;
      Animal.testClassMethod();
      Cat.testClassMethod();
      myAnimal.testClassMethod();
      myCat.testClassMethod();
      myAnimal.testInstanceMethod();
      myCat.testInstanceMethod();
   }
}
```

```
The Animal class method
The Cat class method
The Animal class method
The Cat class method
The Cat instance method
The Cat instance method
```

Пример 5:

```
public class BaseClass {

        public static String staticMethod() {
           return "Base class staticMethod()";
        }

        public String dynamicMethod() {
           return "Base class dynamicMethod()";
        }

        public String dynamicMethod1() {
           return "Base class dynamicMethod1()";
        }
}

public class DerivedClass extends BaseClass {
        public static String staticMethod() {
           return "Derived class staticMethod()";
        }

        public String dynamicMethod () {
           return "Derived class dynamicMethod()";
        }
}

public class Main {

        public static void main(String[] args) {
                BaseClass base = new DerivedClass();
                DerivedClass derivat = new DerivedClass();
                System.out.println(base.staticMethod());
                System.out.println(derivat.staticMethod());
                System.out.println(base.dynamicMethod());
                System.out.println(base.dynamicMethod1());
        }
 }
```

```
Base class staticMethod()
Derived class staticMethod()
Derived class dynamicMethod()
Base class dynamicMethod1()
```

Методи, общи за всички обекти (методи на класа Object)

1. Методи, които подлежат на припокриване: ***equals, hashCode, toString, clone***
   - public boolean **equals**(Object *obj*)

   Контракт на метода: реализира релация на еквивалентност – рефлексивна, симетрична, транзитивна.
   Непротиворечивост на реализацията.
   Сравнението с ***null*** трябва винаги да дава резултат ***false***.
   Кога да се припокрива **equals** – ако равенство на обекти надхвърля рамките на идентичност на обекти.

```
public boolean equals(Object obj)  {
        if(obj == null)
                return false;
        if(this == obj)
                return true;
        if(!(obj instanceof MyClass))
                return false;
        MyClass tmp = (MyClass)obj;
        // all necessary comparions
        return …;
}
```

Допустим начин за проверка на равенство, но не се препоръчва.

```
public boolean equals(MyObject obj)  {
        // all necessary comparions
        return …;
}
```

   - public int **hashCode**()

   Контракт на метода: ако два обекта са равни съгласно **equals** , то за тях методът **hashCode** връща една и съща стойност.
   Непротиворечивост на реализацията.
   Задължително припокриване в клас, който припокрива **equals.**

   - public String **toString**()

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

   - public Object **clone**() throws CloneNotSupportedException

   Контракт на метода: класът трябва да реализира интерфейса ***Cloneable*** и методът да изпълнява следните условия:
   x.clone() != x

```
x.clone().getClass() == x.getClass()
x.clone().equals(x)

public Object clone() {
        try {
                return super.clone();
        }
        catch(CloneNotSupportedException e) {
                throw new Error("Assertion failure");  // Can't happen
        }
}
```

!!! Внимание, ако класът съдържа рефрентни полета.

2. Методи, които не подлежат на припокриване: ***getClass, notify, notifyAll, wait***