Структури от Данни и Обектно-Ориентирано Програмиране
спец. Компютърни Науки
Упражнение №2
04.03.2010г.

ТЕМА: Наследяване и полиморфизъм. Абстрактни класове

Скриване на статични методи

```
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(); // Upcast
                System.out.println(base.staticMethod());
                System.out.println(base.dynamicMethod());
                System.out.println(base.dynamicMethod1());
        }
 }
```

Припокриване на методи с модификатор *private*

```
public class PrivateOverride {
        private void f() {
          System.out.println("private f()");
        }
        public static void main(String[] args) {
          PrivateOverride po = new Derived();
          po.f();  // private  f()
          Derived d = new Derived();
          d.f();  // f() is hidden for Derived!
        }
}

public class Derived extends PrivateOverride {
        public void f() {
           System.out.println("public f()");
        }
}

public class Main {
        public static void main(String[] args) {
           PrivateOverride po = new Derived();
           // po.f();   method f() not visible!
           Derived d = new Derived();
           d.f();
        }
}
```

Припокриване на методи с модификатор *private* и ***final***

```
class WithFinals {
  // Identical to "private" alone:
  private final void f() { System.out.println("WithFinals.f()"); }
  // Also automatically "final":
  private void g() { System.out.println("WithFinals.g()"); }
}

class OverridingPrivate extends WithFinals {
  private final void f() {
          System.out.println("OverridingPrivate.f()");
  }
  private void g() {
          System.out.println("OverridingPrivate.g()");
  }
}
```

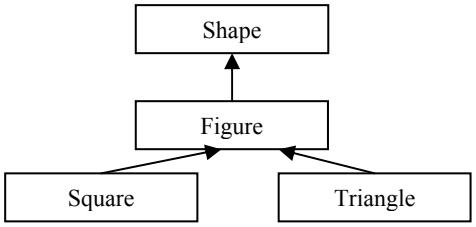
```
class OverridingPrivate2 extends OverridingPrivate {
  public final void f() {
           System.out.println("OverridingPrivate2.f()");
  }
  public void g() {
           System.out.println("OverridingPrivate2.g()");
  }
}

public class FinalOverridingIllusion {
  public static void main(String[] args) {
    OverridingPrivate2 op2 = new OverridingPrivate2();
    op2.f();
    op2.g();
    // You can upcast:
    OverridingPrivate op = op2;
    // But you can't call the methods:
    //! op.f();
    //! op.g();
    // Same here:
    WithFinals wf = op2;
    //! wf.f();
    //! wf.g();
  }
}
```

Абстрактни класове

```
public abstract class Question {
           protected String theText;

           public Question() {
                    theText = "";
           }

           public Question( String text ) {
                    theText = text;
           }

           public abstract void askTheUser();
}

public class YesNoQuestion extends Question {
           public YesNoQuestion( String text ) {
                    super( text );
           }
```

```
        public void askTheUser() {
                System.out.println( theText );
                System.out.println( "YES or NO ...?" );
        }
}


public class FreeTextQuestion extends Question {
        public FreeTextQuestion( String text ) {
                super( text );
        }

        public void askTheUser() {
                System.out.println( theText );
                System.out.println( "Well...?  What's the answer...?" );
        }
}

public class QuestionTest {
        public static void main(String[] args) {
                Question[] questions = getQuestions();
                for( int i = 0; i < questions.length; i++ ) {
                        questions[ i ].askTheUser();  // Polymorphism !!!
                }
        }

        private static Question[] getQuestions() {
                Question[] qs = new Question[ 2 ];
                qs[0] = new YesNoQuestion( "Do you understand polymorphism?" );
                qs[1] = new FreeTextQuestion( "Why is polymorphism good?" );
                return qs;
        }
}
```

Задача: Да се съставят класовете от следната йерархия, като класът *Shape* в основата е абстрактен. Всеки обект на класа *Shape* има цвят – един от фиксиран набор цветове, а също и площ, която се пресмята по подходящ начин. Класът *Figure* също е абстрактен, като обектите от този клас имат и метод за пресмятане на периметъра. Класовете *Square* и *Triangle*, които представят конкретни равнинни фигури, използват обекти на класа *Point* за представяне на съответната фигура.




```
public class Color {

        private String name;

        private Color(String nm) { name = nm; }

        public String toString() { return name; }

        public static final Color
           COLORLESS = new Color("COLORLESS"),
           WHITE = new Color("WHITE"),
           BLACK = new Color("BLACK"),
           BLUE = new Color("BLUE"),
           GREEN = new Color("GREEN"),
           RED = new Color("RED");
}

public class Point implements Cloneable {
        private double x = 0;
        private double y = 0;

        public Point()  {}

        public Point(double x, double y ) {
                this.x = x;
                this.y = y;
        }
```

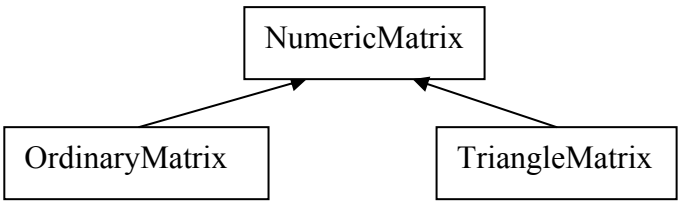
```
// Cloning constructor
public Point(Point p) {
        this.x = p.x;
        this.y = p.y;
}

public double getX () {
        return x;
}

public double getY () {
        return y;
}

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

// Possible but not recommended
public boolean equals(Point p)  {
        return this.x == p.x && this.y == p.y;
}

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

public static Point newInstance(Point p) {
        return new Point(p);
}

public static boolean sameLine(Point a, Point b, Point c)  {
        throw new UnsupportedOperationException();
}
```

```
        public double distanceTo(Point p) {
                throw new UnsupportedOperationException();
        }
}

public abstract class Shape {

        protected Color co = Color.COLORLESS;

        public Shape() {}

        public Color getColor() {
                return col;
        }

        public void setColor(Color c) {
                col = c;
        }

        abstract double area();
}


public class Square extends Figure {

  private Point leftUp;
  private Point rightDown;

  public Square(Point p1, Point p2) {
    this.leftUp = p1;
    this.rightDown = p2;
  }

  public boolean equals (Object q)  {
          throw new UnsupportedOperationException();
  }

  public double len()  {
          throw new UnsupportedOperationException();
  }

  public double diagonal()  {
          throw new UnsupportedOperationException();
  }
```

```
  public double perim() {
        throw new UnsupportedOperationException();
  }

  public double area()  {
        throw new UnsupportedOperationException();
  }
}

public class Triangle extends Figure {

  private Point a, b, c;

  public Triangle(Point a, Point b, Point c) throws InvalidFigure {
    if (Point.sameLine(a, b, c)) throw new InvalidFigure();
    this.a = a;
    this.b = b;
    this.c = c;
  }

  public Triangle(Point a, Point b, Point c, Color col) throws InvalidFigure {
    if (Point.sameLine(a, b, c)) throw new InvalidFigure();
    this.a = a;
    this.b = b;
    this.c = c;
    this.col = col;
  }

  private Triangle() {
    a = new Point(0,0);
    b = new Point(1,0);
    c = new Point(0,1);
  }

  public boolean equals(Object t) {
        throw new UnsupportedOperationException();
  }

  public double ab() {
        throw new UnsupportedOperationException();
  }

  public double ac() {
        throw new UnsupportedOperationException();
  }
```

```
  public double bc() {
        throw new UnsupportedOperationException();
  }

  public double area() {
        throw new UnsupportedOperationException();
  }

  public double perim () {
        throw new UnsupportedOperationException();
  }
}
```

Задача: Да се реализира следната йерархия от класове за представяне на числови матрици, като класът *NumericMatrix* е абстрактен. Елементите на матриците са тип *double*. Двата реални класа се различават по вида на матриците – правоъгълни или триъгълни, което се отразява в различното вътрешно представяне. Целта е двуаргументните операции с матрици, като събиране и умножение, да се извършват успешно независимо от вида на двата аргумента. Ако аргументите са от един и същи вид, то резултатът е от същия вид, а ако са от различни видове – то резултатът е правоъгълна матрица.




```
public abstract class NumericMatrix {

        protected int rows;
        protected int columns;

        abstract double elementAt(int i, int j) throws IllegalArgumentException;

        abstract void setElement(int i,int j,double val) throws IllegalArgumentException;

        public abstract NumericMatrix copyMatrix();

        boolean canAdd(NumericMatrix mat)  {
          return this.rows == mat.rows && this.columns == mat.columns;
        }
```

```
    boolean canMult(NumericMatrix mat)  {
      return this.columns == mat.rows;
    }

    public NumericMatrix() {}

    public NumericMatrix addTo(NumericMatrix mat) throws
                    IllegalArgumentException {
        System.out.println("Here I am in Numeric!");
        if (!canAdd(mat)) throw new
                        IllegalArgumentException("Can't do addition!");
        NumericMatrix result=new OrdinaryMatrix(rows, columns);
        for (int i = 0; i < rows; i++)

          for (int j = 0; j < columns; j++)
            try {
              result.setElement(i,j, this.elementAt(i,j)+mat.elementAt(i,j));
            }
            catch(IllegalArgumentException e) {
            System.out.println(e.toString());
        }
        return result;
    }

    public NumericMatrix multWith(NumericMatrix mat) throws
                        IllegalArgumentException {
        throw new UnsupportedOperationException();
    }

    public void printMatrix() {
        for(int i = 0; i < rows; i++) {
            try {
                for(int j = 0; j < columns-1; j++)
                    System.out.print(elementAt(i, j) + " , ");
                System.out.println(elementAt(i, columns-1));
            }
            catch(IllegalArgumentException e) {}
        }
    }
}
```

За домашна работа:

Задача: Създайте следните класове:
- Person – за описание на хора;
- Student – за описание на студенти;
- Bachelour – студенти-бакалаври;
- Magister – студенти в магистърска програма;
- Freshmann – новоприети студенти(първокурсници)

Организирайте ги в йерархия по подходящ начин и ги снабдете с полета за описание на свойствата им и с методи, описващи поведението им.