Дисциплина: *СД и ООП*
Специалност: *КН 1 курс, 3 и 4 групи*

Тема: *Основни концепции на обектно-ориентираното програмиране*
Седмица: *23.02 - 01.03.2009 г.*

Задача 1: Отсечка наричаме двойката: начало и край – реални числа, като началото е по-малко или равно на края. Да се напише клас на езика Java за работа с отсечки, съдържащ:

1.Декларации на променливи за представяне на отсечка
2.Конструктор с два параметъра за инициализация на отсечка
3.Методи, реализиращи следните операции над отсечки:

- Дължина на отсечка O(H,K): d(O) = K-H
- Сума на две отсечки $O_1 + O_2 = (0, d(O_1) + d(O_2))$
- Произведение на отсечка с реално число O * a = (0, d(O) *a)
- Сравнение на две отсечки $O_1 \leq O_2$, ако $d(O_1) \leq d(O_2)$

2.Метод, който сортира масив от отсечки във възходящ ред

```
public class Otsechka {
	//Data
	private double begin;
	private double end;

	//Constructor
	public Otsechka(double begin,double end) throws RuntimeException {
		if(begin>end) throw new RuntimeException("Invalid data!");
		this.begin=begin;
		this.end=end;
	}

	//Public methods
	public double getBegin() { return begin; }

	public double getEnd() { return end; }

	public double length() { return end-begin; }

	public Otsechka add(Otsechka arg) {
		return new Otsechka(0,this.length()+arg.length());
	}

	public Otsechka mult(double value) {
		return new Otsechka(0,this.length()*value);
	}

	public int compareTo(Otsechka arg) {
		return (int)(this.length()-arg.length());
	}

	public static void sort(Otsechka[] arg) {
		boolean flag=true;
		while(flag) {
			flag=false;
			for(int i=0;i<arg.length-1;i++)
				if(arg[i].compareTo(arg[i+1])>0) {
					flag=true;
					Otsechka temp=arg[i];
					arg[i]=arg[i+1];
					arg[i+1]=temp;
				}
		}
	}
}

public class TestOtsechka {
```

```
	public static void show(Otsechka arg) {
		System.out.println("Begin="+arg.getBegin()+"  End="+arg.getEnd());
	}

	public static void main(String[] args) {
		Otsechka O1=new Otsechka(5,10);
		Otsechka O2=new Otsechka(-2,1);
		Otsechka O3=new Otsechka(0,1);
		Otsechka O4=new Otsechka(7,10);
		Otsechka O5=new Otsechka(-6,-1);

		System.out.println("     Otsechka O1:");
		show(O1);
		System.out.println("     Otsechka O2:");
		show(O2);
		System.out.println("     Otsechka O1+O2:");
		show(O1.add(O2));
		System.out.println("     Otsechka O1*2:");
		show(O1.mult(2));

		Otsechka[] A={O1,O2,O3,O4,O5};
		Otsechka.sort(A);
		System.out.println("     Sorted array:");
		for(Otsechka O:A) show(O);
	}
}
```

Задача 2: Какво извежда всяка от програмите:

1.Програма **Example1**

```
class Parent {
	public Parent() { System.out.println("Constructor Parent()"); }

	public Parent(int n) { System.out.println("Constructor Parent("+n+")"); }
}

class Derived extends Parent {
	private Parent P;

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

	public Derived(int n) {
		super(n);
		P=new Parent(-n);
		System.out.println("Constructor Derived("+n+")");
	}
}

public class Example1 {
	public static void main(String[] args) {
		System.out.println("     new Derived(7):"); Derived D=new Derived(7);
		System.out.println("     new Derived():"); D=new Derived();
	}
}
```

2.Програма **Example2**

```
class Parent {
	public Parent() {}

	public void method1() { System.out.println("Parent method1"); }
	public void method2() { System.out.println("Parent method2"); }
}

class Derived extends Parent {
	public Derived() {}
```

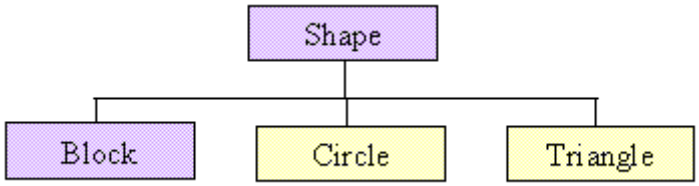
```
	public void method1() {
		super.method1();
		System.out.println("Derived method1");
	}
}

public class Example2 {
	public static void main(String[] args) {
		Parent P=new Parent();
		P.method1(); P.method2();

		Derived D=new Derived();
		D.method1(); D.method2();

		P=D;
		P.method1(); P.method2();
	}
}
```

Задача 3: Да се реализира йерархията от класове за представяне на геометрични фигури в равнината – окръжност, блок, триъгълник:




```
public class Point {
	//Data
	private double x,y;

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

	//Public methods
	public double getX() { return x; }

	public double getY() { return y; }

	public double dest(Point p) { return Math.sqrt(Math.pow(x-p.x,2)+Math.pow(y-p.y,2)); }

	public Object clone() {return new Point(x,y); }

	public boolean equals(Object obj) {
		Point p=(Point)obj;
		return p.x==x && p.y==y;
	}

	public String toString() { return "Point("+x+","+y+")"; }
}

public abstract class Shape {
	public abstract double area();
	public abstract double p();
	public abstract boolean isMember(Point p);
}

public class Block extends Shape {
	//Data
	private Point d1;    //top  left corner
	private Point d2;    //down right corner
```

```
	//Constructors
	public Block(double x1,double y1,double x2,double y2) throws RuntimeException {
		if(x1==x2 || y1==y2) throw new RuntimeException("Invalid data!");
		d1=new Point(x1,y1);
		d2=new Point(x2,y2);
	}

	public Block(Point d1,Point d2) throws RuntimeException {
		this(d1.getX(),d1.getY(),d2.getX(),d2.getY());
	}

	//Public methods
	public double area() {
		Point m=new Point(d1.getX(),d2.getY());
		return m.dest(d1)*m.dest(d2);
	}

	public double p() {
		Point m=new Point(d1.getX(),d2.getY());
		return 2*(m.dest(d1)+m.dest(d2));
	}

	public boolean isMember(Point p) {
		return d1.getX()<=p.getX() && p.getX()<=d2.getX() &&
			d2.getY()<=p.getY() && p.getY()<=d1.getY();
	}

	public Object clone() {
		return new Block((Point)(d1.clone()),(Point)(d2.clone()));
	}

	public boolean equals(Object obj) {
		Block b=(Block)obj;
		return d1.equals(b.d1) && d2.equals(b.d2);
	}

	public String toString() { return "Block="+d1.toString()+","+d2.toString(); }
}

public class TestShape {
	public static void main(String[] args) {
		Circle a=new Circle(new Point(0,0),5);
		System.out.println(a.toString());
		System.out.println("Area="+a.area()+"\nP="+a.p());
		Point p=new Point(2,2);
		System.out.print(p.toString());
		System.out.println((a.isMember(p)?" is in":" is not in")+" circle!");
		System.out.println();

		Block b=new Block(new Point(1,1),new Point(10,15));
		System.out.println(b.toString());
		System.out.println("Area="+b.area()+"\nP="+b.p());
		System.out.print(p.toString());
		System.out.println((b.isMember(p)?" is in":" is not in")+" block!");
		System.out.println();

		Shape[] array={ a, b,(Circle)a.clone() };
		System.out.println("Array of shapes:");
		for(Shape s: array) System.out.println(s.toString());
	}
}
```