Chapter 5 � Inheritance

1. Declare the **GreatPoint** class as a class that extends **MyPoint**, the given class. You should declare in the GreatPoint class the method distance() that calculates the distance of the point from (0,0) and check it using the given application, the class PointDemo.

```
class MyPoint
{
        private double x;
        private double y;
        MyPoint()
        {
        }
        MyPoint(double a, double b)
        {
                x=a;
                y=b;
        }
        public double getX()
        {
                return x;
        }
        public double getY()
        {
                return y;
        }
        public void setX(double xVal)
        {
                x = xVal;
        }
        public void setY(double yVal)
        {
                y = yVal;
        }
}
public class PointDemo
{
        public static void main(String args[])
        {
                GreatPoint gp = new GreatPoint(12,24);
                System.out.println(�The distance of gp from (0,0) is � + gp.distance());
        }
}
```

[solution]

2. Declare the **Person** class that describes a human and the **Student** class that extends Person. The attributes in each instance of type Person should include: name, age and id. The attributes in each instance of type Student should include, in addition to those that inherited from class Person, the attribute average. Each of the two classes should have declared the toString() method. You should check these two classes by instantiating them and invoking the toString method on each one of them. Don�t declare any constructors in the classes Person and Student.

   [solution]

3. Declare the SportCar class, which describes a sport car. The class should extend Car, another class. You should declare, in each one of these classes, at least 2 attributes, at least 2 methods and the toString() method. You should check these classes by instantiating them and checking their toString() method by sending their references to the println method. Each one of the attributes should be private.

   [solution]

4. Declare the following classes:
   The **Rectangle** class**,** that extends Shape, describes a simple rectangle.
   The **Circle** class**,** that extends Shape, describes a simple circle.
   The Shape class is given as follows:

```
public abstract class Shape
{
        public abstract double area();
        public String toString()
        {
                return �The area is � + area();
        }
}
```

   Check your classes using the following application:

```
public class Test
{
        public static void main(String args[])
        {
                Shape vec[] = {new Circle(3), new Rectangle(4,5), new Circle(4), new Circle(8)};
                for(int index = 0; index < vec.length; index ++)
                {
                        System.out.println(vec[index]);
                }
        }
```

}

[solution]

5. Declare the following classes:
The **Manager** class**,** that extends Employee, describes a manager.
The **Clerk** class**,** that extends Employee, describes a clerk.
Define in each one of these classes all the needed methods and constructors and check them using the given application.
The **Employee** class is given as follows:

```
public abstract class Employee
{
        String name;
        double age;
        double hourRate;

        public abstract double salary(double hours);
        public String toString()
        {
                return �name= � + name + � age=� + age + � hourRate=� + hourRate;
        }
}
```

Check your classes using the following application:

```
public class Test
{
        public static void main(String args[])
        {
                Employee vec[] = {new Manager(�Moshe�, 52, 104), new Clerk(�Daniel�, 26, 110), new Manager(�Ramy�,
42, 120), new Manager(�Ronen�, 34, 120)};
                double hours[] = {140,150,130,180};
                double total = 0;
```

```
                    for(int index = 0; index < vec.length; index ++)
                    {
                              total += vec[index].salary(hours[index]);
                                System.out.println(vec[index]);
                    }
                    System.out.println(�The total payment of the employees is � + total);
          }
}
```

[solution]

6. Declare the Rectangle, SportCar, Manager classes as classes that implement the Printable interface and run the given application. The output of this application should be the details of each one of the objects that were instantiated.

```
interface Printable
{
          public void print();
}

public class PrintableDemo
{
          public static void main(String args[])
          {
                    Printable vec[] = {new Rectangle(200,100), new SportCar(�Fiat�, 7892321),
                                              new Rectangle(34,32), new Manager(�Gidi�, 32),
                              new Rectangle(54,10), new SportCar(�Audi�, 4322344),
                              new SportCar(�Mazda�, 4322343), new Manager(�Moshe�, 2344322)};
                for(int index=0; index<vec.length; index++)
                {
                        vec[index].print();
                }
        }
}
```

[solution]

7. The given application should sort 50 objects that represent 50 Rectangles according to their area. In order to make this application run the Rectangle class should extends the Shape abstract class (as given below) and implement the java.lang.Comparable interface (more details can be found in the SDK 1.3 API). Declare the class Rectangle.

```
public class RectangleSort
{
        public static void main(String args[])
        {
                Rectangle vec[];
                vec = new Rectangle[50];
                double randomWidth=0, randomHeight=0;
                for(int index=0; index<vec.length; index++)
        {
            randomWidth = 1000 * Math.random();
            randomHeight = 1000 * Math.random();
            vec[index] = new Rectangle(randomWidth, randomHeight)
        }

        Arrays.sort(vec);

        for(int index=0; index<vec.length; index++)
        {
            System.out.println(vec[index]);
        }
}

abstract class Shape
{
    abstract double area();
    public String toString()
    {
        return �area=�+area();
    }
}
```

[solution]

8. The given application creates 50 objects of the following classes: Rectangle, Circle and EquilateralTriangle. Each of these classes should extend the abstract class Shape which is given partly below. You should declare the three classes and add the missing code in class Shape so the given application will work and sort the shapes according to their area. The classes Circle, Rectangle and EquilateralTriangle should include a new version of the toString() method that returns a string which includes their area and their appropriate descriptive string (one of the follows: Circle, Rectangle or EquilateralTriangle). In defining the toString (in each of the subclasses) you should use the �super� key word in order to call the overridden version.

```
abstract class Shape implements Comparable
{
        abstract double area();
        public int compareTo(Object other)
        {
                //add the missing code




        }

        public String toString()
        {
                return �area=�+area();
        }
```

```
}

public class RectangleSortV2
{
    public static void main(String args[])
    {
        Shape vec[] = new Shape[50];
        for(int index=0; index<vec.length; index++)
        {
            switch((int)(4*Math.random()))
            {
                case 1:
                        vec[index] = new Rectangle(1000*Math.random(), 1000*Math.random());
                        break;
                case 2:
                        vec[index] = new Circle(1000*Math.random());
                        break;
                default:
                        vec[index] = new EquilateralTriangle(1000*Math.random());
            }
        }

        Arrays.sort(vec);
```

```
                for(int index=0; index<vec.length; index++)
                {
                        System.out.println(vec[index]);
                }
        }
}
```

[solution]

9. The given application creates 40 objects of the following classes: Rectangle, Circle and EquilateralTriangle. Each of these classes should extend the abstract class Shape that is given partly below. You should declare the three classes and add the missing code in class Shape so the given application will work and sort the array according to the objects perimeters. The classes Circle, Rectangle and EquilateralTriangle should include a new version of the toString() method that returns a string which includes their area, their perimeter and the their appropriate descriptive string (one of the follows: Circle, Rectangle or EquilateralTriangle). In defining the toString method in each one of the subclasses you should use the �super� key word in order to call the overridden version.

```
abstract class Shape implements Comparable
{
        abstract double area();
        abstract double perimeter();
        public int compareTo(Object other)
        {
                //add the missing code
```

```
          }

          public String toString()
          {
                    return �area=�+area();
          }
}

public class RectangleSortV2
{
          public static void main(String args[])
          {
                    Shape vec[] = new Shape[50];
                    for(int index=0; index<vec.length; index++)
                    {
                              switch((int)(4*Math.random()))
                              {
                                        case 1:
                                                            vec[index] = new Rectangle(1000*Math.random(), 1000*Math.random());
                                                            break;
                                        case 2:
```

```
                                        vec[index] = new Circle(1000*Math.random());
                                        break;
                              default:
                                        vec[index] = new EquilateralTriangle(1000*Math.random(), 1000*Math.random(),
1000*Math.random());
                         }
                    }

                    Arrays.sort(vec);

                    for(int index=0; index<vec.length; index++)
                    {
                         System.out.println(vec[index]);
                    }
               }
}
```

[solution]

10. The given application should create randomly 10 objects of the following classes: **Flafel**, **Pizza** and **Hamburger**. Each of these classes should extend the **Food** class that is given partly below. You should declare the three classes and add the missing code in the given classes so the given application will work and sort the array according to the calories criteria. The Falafel, Pizza and Hamburger classes should have a new version of the toString() method that returns a string which includes their calories and the

their appropriate descriptive string (one of the follows: Falafel, Pizza or Hamburger). In defining the toString method in each one of the subclasses you should use the �super� key word in order to call the overridden version.

```
import java.util.*;

class Food implements Comparable
{
        private long calories;
        Food(long cal)
        {
                calories = cal;
        }
        long getCalories()
        {
                //add the missing code

        }
        public int compareTo(Object other)
        {
                //add the missing code




        }
```

```
          public String toString()
          {
                    return �calories=�+calories;
          }
}

public class Restaurant
{
          public static void main(String args[])
          {
                    Food vec[] = new Food[10];
                    for(int index=0; index<vec.length; index++)
                    {
                              //Creating randomly 10 different objects from the
                              //Falafel, Pizza and Hamburger classes




                    }

                    Arrays.sort(vec);

                    for(int index=0; index<vec.length; index++)
```

```
                {
                        System.out.println(vec[index]);
                }
        }
}
```

[solution]

11. The given application should create randomly 40 objects of the following classes: **MotorolaMobileTelephone**, **NokiaMobileTelephone** and **PanasoniocMobileTelephone**. Each of these classes should extend the **MobileTelephone** class that is given partly below. You should declare the three classes and add the missing code in the given classes so the given application will work and sort the array according to the price criteria. The **MotorolaMobileTelephone**, **NokiaMobileTelephone** and **PanasoniocMobileTelephone** classes should have a new version of the toString() method that returns a string which includes their price and their appropriate descriptive string (one of the follows: **MotorolaMobileTelephone**, **NokiaMobileTelephone** and **PanasoniocMobileTelephone**). In defining the toString method in each one of the subclasses you should use the �super� key word in order to call the overridden version.

```
class MobileTelephone implements Comparable
{
        private double price;
        long getPrice()
        {
                //add the missing code
        }
        public int compareTo(Object other)
        {
                //add the missing code




        }

        public String toString()
        {
                //add the missing code



        }
}

public class MobileTelephonesDemo
{
        public static void main(String args[])
        {
                MobileTelephone vec[] = new MobileTelephone[50];
                for(int index=0; index<vec.length; index++)
                {
                        //Creating randomly 50 different objects from the
                        //MotorolaMobileTelephone, NokiaMobileTelephone
                        //and the PanasonicMobileTelephone




                }

                Arrays.sort(vec);

                for(int index=0; index<vec.length; index++)
                {
                        System.out.println(vec[index]);
                }
        }
}
```

[solution]

12. Declare the **Toaster**, **TV** and **Telephone** classes. Each of these classes should extend the class **Appliance**. The class **Appliance** should implement the Comparable interface. The **Appliance** class should have the following private attributes:
 - voltage:double

- Color:java.awt.Color
- madeIn:String
- price:double

The **Appliance** class should have the following public methods:

+ Appliance(double, java.awt.Color, String, double)
+ getVoltage():double
+ getColor():java.awl.Color
+ getMadeIn():java.util.String
+ getPrice():double
+ setPrice(double):void
+ compareTo(Object):int
+ toString():java.util.String

In the classes that extend the Appliance class you should declare - in each one of them - at least three attributes, three methods and two constructors. Later, code a stand-alone application that instantiated the 3 concrete classes and creates an array (the array type is Appliance) of 40 different objects from the Toaster, TV and elephone classes. The stand-alone application should sort the 40 different objects according to their Price.

[solution]

13. Declare the **PizzaDeluxe**, **PizzaSpecial** and **PizzaWoogy** classes. Each of these classes should extend the **Pizza** class. The **Pizza** class should implement the Comparable interface. The **Pizza** class should have the following private attributes:

- calories:double
- name:String

- price:double

The **Pizza** class should implements the Comparable interface and it should have the following public methods:

+ Pizza(double, String, double)
+ getCalories():double
+ getName():String
+ getPrice():double
+ toString():java.util.String
+ compareTo(Object):int

In the classes that extend the Pizza class you should declare - in each one of them - at least three attributes, three methods and two constructors. Later, code a stand-alone application that instantiates the 3 concrete classes and creates an array (the array type is Pizza) of 40 different objects from the PizzaDeluxe, PizzaSpecial and PizzaWoogy classes. The stand-alone application should sort the 40 different objects according to their Price.

[solution]

14. Declare the **BasketBall**, **FootBall** and **VolleyBall** classes. Each one of them should extend **Game,** another class. Declare the four classes. In a separate class that shell function as a stand-alone application check these classes by instantiating them and putting the references to the new objects within an array that its type is **Object**. The method toString() should be declared in each one of the four classes. A loop should go over each one of the objects and invoke the toString() on each one of them.

[solution]

15. Declare the **JavaProgrammer** class to describe a java programmer. Declare the **JavaDevelopper** class to describe a java developer. The JavaDevelopper should extend the JavaProgrammer class. Declare the **JavaArchitect** class as a class that describes a java architect and extends the JavaDeveloper. Think about the different variables and methods (constructors as well) that should be declared in each one of the classes. Test the classes that you declared using a stand-alone application (another separate class).

    [solution]

16. Declare the **Tree** class to describe a tree. Declare the **Plant** class to describe a plant. The Tree class should extend the Plant class. Declare the **Flower** class as a class that describes a flower and extends the Plant class. Think about the different variables and methods (constructors as well) that should be declared in each one of the classes. Test the classes that you declared using a stand-alone application (another separate class).

    [solution]

17. Declare the **TV** class to describe a television. Declare the **Monitor** class to describe a monitor. The TV class should extend the Monitor class. Think about the different variables and methods (constructors as well) that should be declared in each one of the classes. Test the classes that you declared using a stand-alone application (another separate class).

    [solution]

18. Declare the **School** class to describe a school. Declare the **University** class to describe a university. The University class should extend the School class. Think about the different variables and methods (constructors as well) that should be declared in each one of the classes. Test the classes that you declared using a stand-alone application (another separate class).

    [solution]

19. Declare the **Cake** class to describe a cake. Declare the **Bread** class to describe bread. The Cake class should extend Bread class. Think about the different variables and methods (constructors as well) that should be declared in each one of the classes. Test the classes that you declared using a stand-alone application (another separate class).

    [solution]

20. Declare the **Bus** class to describe a bus. Declare the **Car** class to describe a car. The Bus class should extend the Car class. Think about the different variables and methods (constructors as well) that should be declared in each one of the classes. Test the classes that you declared using a stand-alone application (another separate class).

    [solution]

21. Declare the Train class that describes a train. The class should have the following attributes:
    - locomotive : Locomotive
    - wagons : Wagon[]

The Train class should have the following methods:

+ toString() : java.lang.String

The Locomotive class that describes a locomotive should extends the Wagon class and should have the following attributes:

- numOfDrivers : Person
- engine : Engine

The Wagon class that describes a Wagon should have the following attributes:

- sits : Chair[]
- doors : Door[]
- weight : double

The Person class that describes a person should have the following attributes:

- name : java.lang.String

The Chair class that describes a chair should have the following attributes:

- number : int
- nearWindow : boolean

The Door class that describes a door should have the following attributes:

- height : double
- width : double

Add more attributes, methods and constructors as needed to each of these classes and test them by instantiating the Train class.

[solution]

22. Declare the **Screen** class that describes a video screen. The Screen class should have the following attribute:
    - pixels:Pixel[][]

The Screen class should have the following constructor:

+ Screen(int width, int height)

The **Pixel** class should have the following attribute:

- color:java.awt.Color

Declare the **DrawingScreen** class as a class that extends the Screen class and implements the following interfaces:

The **Drawable** interface that has the following methods:

+ drawRect(int xLocation, int yLocation, int width, int height)

+ drawLine(int xStart, int yStart, int xEnd, int yEnd)

+ drawPoint(int x, int y)

The **Displayable** interface that has the following method:

+ display()

For the moment, the DrawingScreen class you declare should be able to be drawn in textual mode only. You should declare all the mentioned classes and interfaces and check the DrawingScreen class by instantiating it and invoke its methods.

[solution]

For your convenience, hereto the links to all of the chapters.