שעור 3. תכנות מונחה עצמים

ממשקים

למרות ש-JAVA לא תומכת בהורשה מרובה, קיימת ב-JAVA האפשרות להגיע לתוצאות, שדומות לאלה שמושגות באמצעות הורשה מרובה, באמצעות השימוש ב-Interfaces.

כל המתודות שמוגדרות חייבות להיות abstract ועם הרשאת הגישה public !

כל המשתנים חייבים להיות static ו- final !

ממשק מאפשר לראות את אותה המחלקה בכל פעם בכובע שונה, בדוגמא בהמשך: מעגל הוא צורה וגם ניתן להשוואה עם מעגלים אחרים.

Interface היא מחלקה מדומה שיכולה להכיל אך ורק מתודות אבסטרקטיות (abstract) בעלות הרשאת הגישה public, ומשתנים שהם static ו- final.

ניסיון לייצור משתנה שאיננו עומד לכאורה בקריטריונים אלה יתברר כניסיון שתוצאתו בכל זאת משתנה מסוג final ו- static.

המתודות שמוגדרות ב-interface **לא** יכולות להיות: native, static, synchronized, final, private או protected.

ה-interface יכלול מתודות ומשתנים שנרצה שיתממשו במחלקות שיישמו אותו.

כדי שמחלקה תממש interface מסוים שהגדרנו יש לעשות שימוש במילה השמורה implements באופן הבא: בסוף שורת הכותרת של המחלקה יש לרשום implements ואחריה את שמו של ה-interface המיושם.

דוגמא 1:

```
//  interface Sortable -- makes an array of objects sortable
//  by providing a function that can compare them
public interface Sortable {
    public int compare(Sortable other);
}

//  class Employee -- a class for Employees
public class Employee implements Sortable {
    private int SSNum;          //  the social security number
    private String firstname;
    private String lastname;
    private int salary;
    public Employee() {     }
    public Employee(int SSN, String first, String last, int sal) {
        SSNum = SSN;
        firstname = first;
        lastname = last;
        salary = sal;
    }
//  compare function required by Sortable interface
//  orders Employees by social security number
    public int compare(Sortable another) {
        return ((Employee)this).SSNum - ((Employee) another).SSNum;
    }
//  allows us to print an Employee
    public String toString() {
        String emp;
        emp = Integer.toString(SSNum) + "      " + lastname + "   "
                    + firstname + "      " + Integer.toString(salary);
        return emp;
    }
}     // end class Employee
```

```java
//  class Sort -- a class containing static sorting methods that can
//                sort any array of Sortable objects
public class Sort {
// bubble sort
      public static void bubbleSort(Sortable[] arr){
      boolean flag = true;
      for (int i=0; flag && i <arr.length; i++){
            flag = false;
            for (int j=0; j < arr.length-1-i; j++){
                  if(arr[j].compare(arr[j+1]) > 0){
                        flag = true;
                        Sortable temp = arr[j];
                        arr[j] = arr[j+1];
                        arr[j+1] = temp;
                  }
            }
      }
}
// we could add other sorting methods here
}   // end class Sort

// class to show use of Sortable interface

public class SortTest {
    public static void main(String[] args) {
// create an array of Employees, print them, sort them, and print them again

        Employee[] people = new Employee[5];
        people[0] = new Employee(444556666,"Haim","Haimovich",75000);
        people[1] = new Employee(222334444,"Moshe","Moshkovich    ",64000);
        people[2] = new Employee(888776666,"Izhak","Izhaki    ",67000);
        people[3] = new Employee(555667777,"Tomer ","Tamir    ",84000);
        people[4] = new Employee(555668888,"Sarah ","Mordehai ",84000);

        System.out.println("Values before sorting:");
        for (int i = 0 ; i < people.length ; i++)
            System.out.println(people[i]);
        Sort.bubbleSort(people);
        System.out.println("Values after sorting:");
        for (int i = 0 ; i < people.length ; i++)
            System.out.println(people[i]);
    }
}     // end class SortTest
```

דוגמא 2: שימוש בממשק Comparable של java:

```java
/**
* this class represents a 2d point in the plane. <br>
* supports several operations on points in the plane.
*/
public class Point implements Comparable<Point>{

// ******** private data ********
      private double _x, _y;

// ********* constructors ********
   public Point (double x1, double y1) {
      _x = x1;
      _y = y1;
   }
/** copy constructor:
 1)here a direct access to a class member is performed,
 this will be done only in a constructor to achieve an identical copy
```

```
 2) using a call to another constructor code is not written twice
 **/
   public Point (Point p){
       this(p._x, p._y);
   }

    // ********** public methodes *********
   public double x(){return _x;}
   public double y(){return _y;}
   public int compareTo(Point p){
         int ans;
         if(this.distance() < p.distance()) ans = -1;
         else if (this.distance() > p.distance()) ans = 1;
         else ans = 0;
         return ans;
   }

   /** @return a String contains the Point data*/
   public String toString(){
      return "name: " + this.getClass() +"[" + _x + "," + _y+"]";
   }

   /**    logical equals
   @param p other Object (Point).
   @return true iff p instance of Point && logicly the same) */
   public boolean equals (Point p){
        return p!=null && p._x == _x && p._y==_y;
   }
   public double distance (){
      return Math.sqrt(Math.pow(_x,2)+Math.pow(_y,2));
   }
}//end class Point

import java.util.Arrays;
public class TestCompPoint {
      public static void main(String[] args) {
            Point p[] = new Point[5];
            p[0] = new Point(1, 2);
            p[1] = new Point(1, 1);
            p[2] = new Point(0, 0);
            p[3] = new Point(4, 2);
            p[4] = new Point(2, 2);
            Arrays.sort(p);
            for (int i=0; i<p.length; i++){
                  System.out.println(p[i].toString());
            }
      }
}
```

Output:

```
name: class Point[0.0,0.0]
name: class Point[1.0,1.0]
name: class Point[1.0,2.0]
name: class Point[2.0,2.0]
name: class Point[4.0,2.0]
```

Interfaces לצורך סימון

ניתן להגדיר Interface שלא מוגדרת בו אף מתודה ואף קבוע, ושמטרתו היחידה לסמן את המחלקה שמיישמת אותו. סימון מחלקה באמצעות Interface שמיושם בהגדרתה נעשה אודות למילה השמורה instanceof. באמצעות instanceof ניתן לדעת אם אובייקט מסוים נוצר ממחלקה שמיישמת Interface מסוים ולטפל בו בהתאם.
משפט התנאי:
if (referenceToAnObject instanceof InterfaceName)
. . .
יהיה true אם האובייקט נוצר ממחלקה שבהגדרתה יושם ה-Interface ששמו צוין. אופן השימוש המלא במילה השמורה instanceof יוסבר בהמשכו של הפרק.

אחת הדוגמאות ל-Interface שמשמש לסימון מחלקות הוא ה-Interface ששמו: Cloneable.

שינוי טיפוס של references (Casting between Class Types)

כפי שכבר ראינו, אין כל בעיה לאחסן reference לאובייקט במשתנה מטיפוס המחלקה שהורישה למחלקה שממנה האובייקט נוצר.

```
Shape sh = new Circle("green", false, 1);
```

באופן דומה, ראינו גם כי אין כל בעיה לשלוח למתודה שמצפה לקבל reference מטיפוס מסוים reference לאובייקט מטיפוס מחלקה שיורשת מאותו טיפוס מסוים.

באופן כללי ניתן לומר כי ביצוע הפעולות שהפוכות לפעולות שתוארו לעיל איננו אפשרי. לא ניתן לאחסן בתוך משתנה מטיפוס מסוים reference מטיפוס שהוריש לטיפוס של אותו משתנה. באופן דומה, לא ניתן לשלוח למתודה reference מטיפוס שהוריש לטיפוס הפרמטר של אותה מתודה.

ביצוע casting ל-reference מתאפשר אך ורק אם תוצאות הבדיקה הבאה true:

```
referenceToObject  instanceof  class/InterfaceName
```

ערכו של הביטוי הלוגי הנ"ל true, אך ורק אם ה-reference ממלא את אחד התנאים הבאים:

1. ה-reference הוא לאובייקט מטיפוס המחלקה ששמה מצוין
2. ה-reference הוא לאובייקט מטיפוס מחלקה שיורשת מהמחלקה ששמה מצוין
3. ה-reference הוא לאובייקט מטיפוס מחלקה שמיישמת את ה-interface ששמו מצוין

בכל אחד מהמקרים הנ"ל ערכו של הביטוי יהיה true, ולכן, ניתן יהיה לבצע ל-reference האמור casting כך שישנה את טיפוסו ויהיה מטיפוס המחלקה או ה-interface ששמם צוין בפקודת ה-instanceof.