**תרגיל 11 לעבודה עצמית – אובייקטים, מחלקות ואוספים**

**שאלה 1 כתוב מחלקה בשם Rectange**

המלבן מוגר ע"י שני משתני עצם:

double w - רוחב המלבן

double h - גובה המלבן

צלעותיו המלבן מקבילים לצירים

המחלקה צריכה להכיל שיטות הבעות:

- בנהי רגיל: public Rectangle(double w, double h){…}
- בנאי מעתיק : public Rectangle(Rectangle r){…}
- שיטה המחשבת היקפו של המלבן public double perimeter(){…}
- שיטה המחשבת שיטחו של המלבן public double area(){…}
- שיטת toString()
- שיטה בוליאנית שמקבלת נקודה ומחזירה true אם הנקודה נמצאת בתוך מלבן:
  public boolean contains(Point p)
- שיטה להשוואת שני מלבנים על פי שטחים שלהם public int compare(Rectangle r){…}
  השיטה מחזירה 1- אם שיטחו של מלבן מקורי קטן משטחו של r, מחזירה 1 אם שיטחו של מלבן מקורי גדול משטחו של r, ומחזירה 1 אם שטחים שווים.
- כתוב פונקציה סטטית שמקבלת מערך של מלבנים וממיינת אותו מקטן לגדול:
  public static void sort(Rectangle[] rectArr){…}
  הדרכה: השתמש באחד המיונים הידועים לך: Selection Sort, Bubble Sort. השוואת מלבנים נעשה ע"פ שטחים.
- כתוב פונקציית main במחלקה אחרת (בשם Plane) הבודקת אל כל הפונקציות הנ"ל.

**שאלה 2:**

בשאלה זו נשתמש במלבן שמוגדר שאלה קודמת ונוסיף שיטה למחלקה PointContainer:

הוסיפו שיטה שמקבלת מערך של מלבנים ומחזירה את הנקודה שמוכלת במספר הגדול ביותר של מלבנים.

הדרכה: אם יש (מספר נקודות שמוכלות במספר מקסימלי של המלבנים – החזירו אחת מהן), ניתן להניח שמערך המלבנים אינו ריק.

```
Point maxIn(Rectangle[] arr){...}
```

**שאלה 3**

כתבו פונקציה סטטית שמקבלת ArrayList ומחזירה את שם המחלקה שממנה יש הכי הרבה אובייקטים במערך.

הנחה: בהגדרה של list לא עשינו המרה לטיפוס מסוים.

```
String maxClasses(ArrayList list)
```

הדרכה: השיטה getClass().getName() מחזירה מחרוזת עם שם המחלקה של כל אובייקט.

**שאלה 4**

כתוב מחלקה בשם Line שתייצג קו ישר העובר בין שתי נקודות במישור Point p1 ו- Point p2.
השתמש במחלקת Point המצורפת.

למחלקה שני משתנים:
Point p1 - נקודה ראשונה
Point p2 – נקודה שנייה

כתוב:

- בנאי: (Constructor)
  public Line (Point p1, Point p2)
  ניתן להניח כי $p1 \neq p2$, כלומר ההנחה היא שהקלט תקין.

- בנאי מעתיק: (Copy Constructor)
  public Line (Line line)

- כתוב פונקציה בוליאנית שמקבלת נקודה ומחזירה אמת אם הנקודה נמצאת על הקו, אחרת היא מחזירה שקר:
  public boolean isOnLine(Point p)

- כתוב פונקציה בוליאנית שמקבלת נקודה ומחזירה אמת אם הנקודה נמצאת מעל הקו, אחרת היא מחזירה שקר:
  public boolean isAbove(Point p)

- כתוב פונקציה public String toString() המציגה את הקו כמחרוזת.

- כתוב פונקציה להשוואת שני קווים. הפונקציה מחזירה -1 אם ישרים מקבילים, 0 אם הם מתלכדים ומחזירה 1 אם הם נחתכים בנק' אחת.
  public int intersection (Line line)

- כתוב פונקציית main במחלקה אחרת (בשם Plane) הבודקת אל כל הפונקציות הנ"ל.

- הוסיפו למחלקה Line אפשרות לחישוב מספר קווים שיוצרים אותם ב-main .

**שאלה 5**

הוסיפו ל-PointContainer שיטה שמחשבת מספר נקודות הנמצאות במעגל יחידה. נקודה נמצאת בתוך מעגל יחידה כאשר מרחקה עד ראשית הצירים קטן או שווה 1.
public int numPointsInCirc1(){…}

**שאלה 6**

הוסיפו ל-PointContainer שיטה שמחזירה מערך שמיכל את כל הנקודות הנמצאות במעגל יחידה.
public Point[] allPointsInCirc1(){…}

```
//////////////////////////////////// Point ////////////////////////////
/** this class represents a 2d point in the plane. */
public class Point {
// ******** private data members ********
        private double _x;  // we "mark" data members using _
        private double _y;

// ********* constructors ********
  public Point (double x1, double y1) {_x = x1; _y = y1; }
  public Point (Point p) {_x = p.x(); _y = p.y(); }

  // ********** public methods *********
  public double x() {return _x;}
  public double y() {return _y;}

  /** @return the L2 distanse */
  public double distance (Point p)  {
    double temp = Math.pow (p.x() - _x, 2) + Math.pow (p.y() - _y, 2);
    return Math.sqrt (temp);
  }

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

 /** logical equals: return true iff p instance of Point && logically the same) */
  public boolean equals (Object p) {
          return p!=null && p instanceof Point &&
        ((Point)p)._x == _x && ((Point)p)._y==_y;
  }
}// class Point

//////////////////////////////////// PointContainer ////////////////////////////
/** this class represent a collection of Points. */
public class PointContainer {
        // *** data members ***
        public static final int INIT_SIZE=10;  // the first (init) size of the set.
        public static final int RESCALE=10;   // the re-scale factor of this set.
        private int _sp=0;
        private Point[] _points;
        /*** Constructors: creates a empty set ***/
        public PointContainer(){
                _sp=0;
                _points = new Point[INIT_SIZE];
        }
```

```
	/** returns the actual amount of Point contains in this set */
	public int size() {return _sp;}

	/** add a Point to this collection */
	public void add (Point p){
		if (p != null){
			if(_sp==_points.length) rescale(RESCALE);
			_points[_sp] = new Point(p);  // deep copy semantic.
			//_points[_sp] = p;		// copy reference, (not deep).
			_sp++;
		}
	}

	/** returns a reference to the Point at the index, (not a copy) */
	public Point at(int p){
		if (p>=0 && p<size()) return _points[p];
		return null;
	}

	/************** private methods *************/
	private void rescale(int t) {
		Point[] tmp = new Point[_sp+t];
		for(int i=0;i<_sp;i++) tmp[i] = _points[i];
		_points = tmp;
	}
}
```