המרכז האוניברסיטאי אריאל בשומרון

מבחן במבוא לתכנות מונחה עצמים

סמסטר א' מועד א' תשע"ג תאריך: 15.07.2013
מס' קורס 2-7026910-2

מרצה: גב' אליזבט איצקוביץ

חומר עזר מותר לשימוש: אין חומר עזר

אין להוציא את השאלון בסוף המבחן - יש להשאירו במחברת הבחינה

משך המבחן : 150 דקות

הוראות כלליות :

1. תשובות יש לכתוב במחברת המצורפת בלבד.
2. נא לכתוב בכתב ברור ומסודר.
3. במבחן 5 שאלות, יש לענות על כל השאלות. משקל כל שאלה 20 נקודות.
4. תשובות מסורבלות או ארוכות מדי לא יזכו בניקוד מלא.
5. אם לא נאמר אחרת ניתן להשתמש בחומר המצורף לבחינה בפתרון השאלות, מעבר לכך בהחלט ניתן לפתור שאלה בעזרת שאלה אחרת.

הקפידו על טהור הבחינה!

בהצלחה!

שאלה 1 (20 נקודות, כל שאלה 2 נקודות))

1. תן הגדרה למחלקה.
2. מהו התפקיד של constructor
3. תן הגדרה ל-protected.
4. מהו ההבדל בין class לבין abstract class?
5. מתי חייבים להגדיר default constructor?
6. הסבר מושג של serialization.
7. מהו ההבדל בין שיטה לפונקציה סטטית?
8. מהי משמעות של מילה שמורה this?
9. הסבר מושג של הורשה.
10. מהו מצב של deadlock?.

שאלה 2 (20 נקודות)

א) (5 נקודות)

הסביר בקצרה מהם תכונות של TreeSet.

ב) (15 נקודות)

בשאלה זה נבנה משחק כדורסל.

מחלקת כדור (**Ball**) מכילה את נקודה שבה נמצא הכדור על שדה כדורסל (מישור) - משתנה מטיפוס Point, בנאי ופונקציה toString().

הכדור יכול לזוז קדימה, אחורה, ימנה ושמאלה, לשם כך המחלקה צריכה להכיל 4 שיטות:

moveLeft – השיטה מקבלת מרחק להזזת כדור שמולה, השיטה משנה (מקטינה) רק את שעור x של הכדור.

moveRight – השיטה מקבלת מרחק להזזת כדור ימינה, השיטה משנה (מגדילה) רק את שעור x של הכדור.

moveForward –השיטה מקבלת מרחק להזזת כדור קדימה, השיטה משנה (מגדילה) רק את שעור y של הכדור.

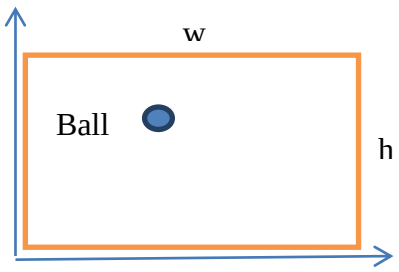
moveBack – השיטה מקבלת מרחק להזזת כדור אחורה, השיטה משנה (מקטינה) רק את שעור y של הכדור.

מחלקת כדורסל (**Basketball**) יורשת ממחלקת Ball.

מחלקת Basketball מכילה גודל של שדה כדורסל: אורך w ורוחב h.

מחלקת Basketball מגבילה את תנועת הכדור, הכדור לא יכול לצאת מהשדה. המחלקה צריכה להכיל בנאי, בנאי מעתיק, שיטת toString() וכל שיטות התנועה.

השתמש בהורשה!



<u>שאלה 3</u> (20 נקודות)

א) (10 נקודות)

הוסף לשיטות מחלקת Container אפשרות לזרוק חריגה בשם ContainerException בכל שיטות שעלולות לעוף בגלל הפרמטרים הלא תקינים. שנה את השיטות האלו לפי דרישה חדשה של זריקת חריגה.

<u>הדרכה</u>: עליך לכתוב מחלקה פשוטה ביותר של ContainerException.

ב) (10 נקודות)

הוסיפו למחלקת אוסף (Container) שיטה שמקבלת אובייקט מסוג Comparator (ראה את הקוד המצורף) ומחזירה את האיבר השני בגודלו של האוסף לפי שיטת השוואה של ה- Comparator:

```
public Object  maxSecond (Comparator comp)
```

<u>שאלה 4</u> (20 נקודות)

א) (10 נקודות)

הפך את מחלקת Container למחלקה גנרית.

ב) (10 נקודות)

הוסף לשיטות מחלקת Container שיטה שמחזירה אובייקט מטיפוס איטרטור ומכילה מחלקה אנונימית המממשת איטרטור:

```
public Iterator containerIterator(){. . .}
```

השלימו את השיטות hasNext() ו- next().

<u>שאלה 5</u> (20 נקודות)

כתוב שלוש מחלקות: Storage, Counter ו- Printer.

מחלקת Storage מחזיקה מספר שלם.

מחלקת Counter צריכה ליצור thread, שמתחיל ספירה מאפס (0,1,2,3, . . .) ושומר כל ערך במחלקת Storage.

מחלקת Printer צריכה ליצור thread, שקורא את הערכים הנמצאים באובייקט של מחלקת Storage ומדפיס אותם.

כתוב תכנית (main) שיוצרת אובייקטים של שלוש המחלקות ומפעילה את ה-threads של מחלקות Counter ו-Printer. כל המספר צריך להיות מודפס רק פעם אחת ובאותו סדר שה-Counter יוצר אותו.

<u>הדרכה</u>: השתמש בסנכרון רלוונטי.

הקוד המצורף לבחינה

```
////////////////////////////////////////////////////////////////
public class Container {
	// *** 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 Object[] _data;

/************ Constructors ************/
	public Container(){
		_sp=0;
		_data = new Object[INIT_SIZE];
	}
	public Container(Container other) {  // copy constructor
		this();
		for(int i=0;i<other.size();i++)
			this.add(other.at(i));
	}

	/** return true is this collection is empty, else return false. */
	public boolean isEmpty() {return _sp==0;}

	/** add an Object to this set */
	public void add (Object p){
		if(_sp==_data.length) rescale(RESCALE);
		_data[_sp] = p;  // shallow copy semantic.
		_sp++;
	}

	/** returns the actual amount of Objects contained in this collection */
	public int size() {return _sp;}
	/** return the index of the first object which equals ob, if none returns -1 */
	public int get(Object ob) {
		int ans=-1;
		for(int i=0;i<size();i=i+1)
			if(at(i).equals(ob)) return i;
		return ans;
	}
	/************** private methodes *************/
	private void rescale(int t) {…}
} // class Container
```

```
/************* Interface Comparator *************/
public interface java.util.Comparator {
        public int compare(Object arg1, Object arg2);
}

/************* Interface Comparable *************/
public interface java.util. Comparable {
        public int compareTo(Object o);
}

/** 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
```