תכנות מונחה עצמים – מטלה 1- ירושה וממשקים

חלק 1 – ירושה INHERITANCE

בתרגיל זה נתאמן בשימוש בירושה, את התרגיל ניתן להגיש בזוגות (או לבד).

בתרגיל זה נכיר צבים חביבים, ונשתמש בהורשה ופולימורפיזים כדי להרחיב את משפחת הצבים .

המחלקה SimpleTurtle מגדירה עבורנו צב רגיל. יצירה של צב מציירת דמות של צב במרכזו של מסך גרפי. המסך הגרפי נוצר עם יצירתו של צב בפעם הראשונה. לצב יש יכולת תנועה. הוא יכול להסתובב סביב עצמו ימינה ושמאלה בכל מעלה שלמה ולפנות לכיוונים שונים. צב חדש נוצר כשהוא פונה כלפי מעלה. צב יכול גם לנוע קדימה אל הכיוון אליו הוא פונה לכל מרחק נתון.

לצב יש זנב. הזנב יכול להיות מורם או מורד. אם הזנב מורד והצב נע קדימה הצב משאיר לאורך מסלולו עקבות בצורה של קו על המסך הגרפי. אם הזנב מורם הצב לא משאיר עקבות. צב גם ניתן להסתרה וגילוי מחדש. בין אם הצב גלוי ובין אם הוא מוסתר הוא מבצע את כל הפעולות באותו האופן. כלומר, הוא יכול להסתובב, להתקדם קדימה, להשאיר עקבות כאשר זנבו מורד, או לא להשאיר סימן כאשר זנבו מורם.

לממשק הציבורי של המחלקה SimpleTurtle המאפיינים והיכולות הבאים:

```
SimpleTurtle (); // construct simple turtle

public void show (); // show yourself

public void hide (); // hide yourself

public void tailDown (); // lower the tail

public void tailUp (); // lift the tail

public void turnLeft (int degrees); // turn left in the given degrees

public void turnRight (int degrees); // turn right in the given degrees

public void moveForward (double distance); // advance forward in the given distance
```

כדי להשתמש במחלקה SimpleTurtle יש להוריד מאתר הקורס אל המחשב שלכם את החבילה **Turtle.jar**, שמאגדת בתוכה מספר קבצים הנדרשים לעבודה עם הצב, יש להגדיר פרויקט ולייבא (import) את החבילה כך שנוכל להשתמש בה בפרויקט (ראה תמונה), דוגמא פשוטה מאוד לקובץ main שמשתמש בחבילה ניתן למצוא בקובץ: **TestTurtle.java**.

לכתיבת החלק הראשון עקבו אחרי המשימות הבאות.

משימה ראשונה – הגדרה של צבים עם תכונות שונות ע"י הורשה

לא כל הצבים נוצרו שווים. יש חכמים, יש מוכשרים, ולא עלינו, מוזרים. הוסיפו מחלקות שיגדירו את הצבים הבאים:

- *צב חכם* (SmartTurtle) – צב חכם, מעבר להיותו צב רגיל לכל דבר ועניין, מבין גם משהו בגיאומטריה: הוא יודע לצייר ריבוע באורך נתון ומצולע משוכלל בעל מספר צלעות נתון באורך נתון. כתבו את המחלקה SmartTurtle, שימרו אותה בקובץ בשם SmartTurtle.java במחיצת העבודה, והוסיפו לה את שתי השיטות הבאות.

public void drawSquare (double size); // draw a square in the given size

public void drawPolygon (int sides, double size);

// draw a polygon in the given sides and size

שימו לב: כיוון שהזויות העוברת כפרמטר בשיטות turnLeft ו- turnRight היא זווית שלמה עלולה להתעורר בעיה במקרה של פוליגון משוכלל עם זווית לא שלמה. <u>התעלמו מהבעיה</u>. דאגו רק שהצב החכם יצייר נכון פוליגונים בעלי זווית שלמה.

- *צב שיכור* (DrunkTurtle) – צב שיכור הוא צב רגיל ששתה מעט וכתוצאה מכך קשה לו קצת ללכת. כשהוא מתבקש לנוע קדימה למרחק x הוא מבצע את הפעולות הבאות:
  - הוא מתקדם למרחק מקרי בין 0 ל- x2.
  - פונה בזווית מקרית בין +30 ל- -30 מעלות.

כתבו את המחלקה DrunkTurtle ושימרו אותה בקובץ בשם DrunkTurtle.java במחיצת העבודה. שנו בה את הדרוש שינוי.

- *צב מקרטע* (JumpyTurtle) – צב מקרטע הוא צב חכם מקרטע: כאשר הוא מתקדם הוא הולך ומנתר לסירוגין. התוצאה היא שכאשר זנבו מורד הוא משאיר קו מקווקו. כתבו את המחלקה JumpyTurtle ושימרו אותה בקובץ בשם JumpyTurtle.java במחיצת העבודה שלכם. שנו בה את הדרוש שינוי.

שימו לב: כיוון שצב מתקדם על פני סריג של נקודות, הצב לא יכול להתקדם תמיד למרחק הנדרש בדיוק נמרץ (למה?). התקדמות למרחק קצר עלולה ליצור אי דיוק גדול יחסית (למה?). יוצא איפה שאם ההתקדמות של צב מקרטע תשבר לרצף ארוך של פסיעות ודילוגם קטנים, אי הדיוק יצטבר והצב עלול להתקדם בפועל למרחק שונה מהנדרש. <u>התעלמו מהבעיה</u>. שיברו את המרחק אליו מתבקש הצב המקרטע להתקדם למספר פסיעות וניתורים קטן, והניחו תמיד שהצב יידרש להתקדם למרחקים גדולים.

חשוב ביותר: על כל הצבים מכל הסוגים <u>לא</u> להשאיר מאחריהם עקבות כאשר הזנב שלהם מורם.

משימה שנייה – ניהול צבא של צבים באמצעות פולימורפיזים

כתבו תוכנית בשם Army.java שתנהל צבא (מערך) של 5 צבים.

- בשלב ראשון אפשרו למשתמש לבחור את צבא הצבים כרצונו. הציגו לפניו את התפריט שלמטה וקבלו את בחירתו עבור כל אחד מחמשת הצבים בנפרד. אפשרו לו לבחור כל תערובת של צבים. להלן התפריט:

Choose the type of a turtle:

1. Simple
2. Smart
3. Drunk
4. Jumpy

- בשלב שני צרו את הצבים הנדרשים וקדמו אותם <u>יחד</u> שלב אחר שלב על פני השלבים הבאים:

1. הורדת זנב
2. צעידה קדימה למרחק של 100
3. פניה של 90 מעלות ימינה
4. צעידה קדימה למרחק של 60
5. לכל מי שיודע לצייר מצולע, ציור של מצולע בן 6 צלעות באורך של 70
6. העלמות

שימו לב, על הצבים להתקדם כולם יחד. אף צב לא יכול לעבור לשלב גבוה יותר בטרם גמרו כל הצבים האחרים את השלב הקודם.

- הריצו את התוכנית ובדקו אותה. הסבירו בתיעוד את השימוש בפולימורפיזם. הסבירו את השימוש ב- casting. **שימו לב**, אל תשתמשו ב- casting אלא אם כן הוא הכרחי.

חלק 2 – ממשקים – INTERFACE

א) יש להגדיר מחלקה Complex המציגה מספרים מרוכבים. המחלקה מכילה תכונות ושיטות הבאות:

(a - חלק ממשי של מספר מרוכב, b – חלק מדומה של מספר מרוכב)

```
public class Complex{
        double  a,  b;
// בנאי
   public Complex(double a, double b){…}
```

♦ שיטה לחישוב מודול של מספר מרוכב $z = (a, b)$, לפי הנוסחה $//|z| = \sqrt{a^2 + b^2}$

```
   public  double module(){ . . }
```

♦ שיטה לחישוב סכום של שני מספרים מרוכבים $z_1 = (a_1, b_1),\ z_2 = (a_2, b_2)$ לפי הנוסחה:// $//z_1 + z_2 = (a_1 + a_2, b_1 + b_2)$

```
  public void add (Complex z){ . . .}
```

♦ שיטה לחישוב כפל של שנה מספרים מרוכבים $z_1 = (a_1, b_1),\ z_2 = (a_2, b_2),$ לפי הנוסחה:// $//z_1 \cdot z_2 = (a_1a_2 - b_1b_2,\ a_1b_2 + a_2b_1)$

```
   public void mul (Complex z){ . . .}
```

♦ נגדיר השוואה של שני מספרים מרוכבים לפי המודולים שלהם: מספר z_1 קטן ממספר z_2 אם $|z_1| < |z_2|$, מספרים z_1 ו- z_2 שווים אם שווים המודולים שלהם $|z_1| = |z_2|$.

שיטה להשוואה של שני מספרים מרוכבים מחזירה:

-1, כאשר מספר Z גדול ממספר שממנו השיטה מופעלת,

0, כאשר מספר Z שווה ממספר שממנו השיטה מופעלת,

+1, כאשר מספר Z קטן ממספר שממנו השיטה מופעלת,

```
   public int compare(Complex z){ . . .}
```

ב) נתון ממשק **Sortable**

```
interface Sortable {
      int compare(Object left, Object right);
      Object valueAt(int position);
      void setValue(Object value, int position);
      int size();
} // interface Sortable
```

יש לכתוב מחלקה **SortComplex** אשר מממשת את הממשק הנתון. המחלקה מכילה מערך של מספרים מרוכבים, והבנאי מקבל מערך של מספרים מרוכבים כפרמטר:

```
private Complex[] cArr;

public sort (Complex[] c){……}
```

ג) יש להגדיר מחלקה **ObjectSort** המכילה שתי שיטות:

- שיטה סטטית `public static void sort(Sortable item)`
  המקבלת עצם ממחלקה אשר מממשת את הממשק **Sortable** וממיינת אותו.
- שיטה סטטית `public static boolean checkSort(Sortable item)`
  המקבלת עצם ממחלקה אשר מממשת את הממשק **Sortable** והמחזירה **true** כאשר המערך ממוין, אחרת היא מחזירה **false**.

ד) לאחר מכן יש להגדיר מחלקת `TestSort` עם main אשר יוצרת עצם ממחלקה **SortComplex** וממיינת אותו באמצעות שיטה `SortObject.`***`sort`*** ובודקת האם המערך ממוין.

נספח:

1) מיון הכנסה של מערך של מספרים שלמים:

```
public static void insertionSort1(int[] arr){
      for (int i=1; i <arr.length; i++){
            int j = i;
            while (j>0 && arr[j]<arr[j-1]){
                  swap(arr, j, j-1);
                  j = j-1;
            }
      }
}
```

2) פונקציה הבודקת האם מערך של מספרים שלמים ממוין:

```
public static boolean checkSort(int[] arr){
      boolean ans = true;
      for (int i=0; ans&&i< arr.length-1; i++){
            if(arr[i] > arr[i+1]){
                  ans = false;
            }
      }
      return ans;
}
```

3) תוכנית TEST:

```
public class TestSort {
      public static void main(String[] args) {
            Complex []c = new Complex[10];
            for(int i=0; i<c.length; i++){
                  double a = Math.random()*c.length;
                  double b = Math.random()*c.length;
                  c[i]= new Complex(a,b);
            }
            SortComplex sc = new SortComplex(c);
            SortObject.sort(sc);
            System.out.println("is sorted: "+SortObject.checkSort(sc));
            for(int i=0; i<c.length; i++){
                  System.out.println(((Complex)sc.valueAt(i)).module());
            }
            System.out.println();
      }
}
```