

שאלה 1 (32 נקודות)

תוכנה לשרטוט הנדסי מייצרת הנחיות לתווין (Plotter) בצורת רשימה של קשתות.

קשת (Arc) היא קטע ישר (Segment), קשת מעגלית (Circular) או קו פוליגונאלי (Polyline).

כל קשת באשר היא מאופיינת ע"י צבע עט ועובי עט. קשתות נמתחות בין צמתים (Node's).

- **קטע ישר** נמתח בין צומת התחלה לצומת סיום.
- **קשת מעגלית** נמתחת בין צומת התחלה לצומת סיום סביב צומת המרכז, עם כוון השעון.
- **קו פוליגונאלי** נתון ע"י רשימת צמתים והוא מחבר בקטע ישר כל צומת (עד אחד לפני האחרון ברשימה) לצומת הבא ברשימה.

נתון להלן הקוד המתאר את המחלקה Node, את המחלקה האבסטרקטית Arc, וחלקים מהמחלקות שמרחיבות אותה.

```
public class Node {
    private double x,y;

    public Node(double x, double y) {this.x = x; this.y = y;}

    public double getx() {return x;}
    public double gety() {return y;}
    public String toString() {return "x = " + x + " y =" + y;}
    public double distanceToNode (Node anotherNode) {
        // השרות מחזיר את המרחק בין הצומת הנוכחי ובין anotherNode
    }
}
```

```
abstract class Arc{
    protected double width;
    protected Color color;

    public Arc(double width, Color color) {
        this.width = width;
        this.color = color;
    }

    public double getWidth() {return this.width;}
    public Color getColor() {return this.color;}
}
```

```
public class Segment extends Arc{
    private Node start, end;

    public Segment(double width, Color color, Node start, Node end)
    {...}

    public Node getStartNode() {return start;}
    public Node getEndNode() {return end;}

    // copy constructor: creates a duplicate of the parameter
    public Segment(Segment anotherSegment) {//ראו טעיף ב'}
}
}
```

```
public class Polyline extends Arc {
    private List<Node> nodeList;

    public Polyline(double width, Color color, List<Node> nodeList){
        super(width, color);
        this.nodeList = new ArrayList<Node>(nodeList);
    }

    public List<Node> getNodeList() {return nodeList;}
}
}
```

```
public class Circular extends Arc{
    private Node start, end, center;

    public Circular(double width, Color color, Node start, Node end,
                    Node center)
    {...}

    public Node getStartNode() {return start;}
    public Node getEndNode() {return end;}
    public Node getCenterNode() {return center;}

    // compute the angle of the circular arc (e.g., for half a
    // circle getRadians() will return PI)
    public double getRadians() {
        // אין צורך לחמש שרות זה
    }
}
}
```

לנוחותכם, מצורף בעמוד האחרון חלק מתיעוד המנשק List

א. (10 נקודות)

אנו מעריכם את עלות הדיו להדפסה של רשימת קשתות `arcList`, ע"י סכימה של מכפלת אורכי הקשתות בעובי שלהן. ממשו את השרות `computeInkCost` של המחלקה `Plotter`. השרות מקבל רשימת קשתות ומחזיר את עלות ההדפסה.

תזכורת: אורך קשת בזווית α על מעגל שרדיוסו R הוא $\alpha \cdot R$ (כאשר 2π מייצג קשת בזווית 360 מעלות)

אם לצורך המימוש ברצונכם להוסיף או לשנות קוד של מחלקות אחרות, עשו זאת בעמוד הבא.

```
public class Plotter {  
  
    // ...  
  
    public static double computeInkCost(List<Arc> arcList) {  
        double inkCost = 0;  
  
        for (Arc arc: arcList)  
            inkCost += arc.computeInkCost();  
  
        return inkCost;  
    }  
}
```

קוד של מחלקות אחרות:

// Additions to Arc.java

```
abstract class Arc{
    // ...

    public abstract double computeInkCost() ;
}
```

// Additions to Segment.java

```
public class Segment extends Arc{
    // ...

    public double computeInkCost() {
        return width*getNode().distanceToNode(getEndNode());
    }
}
```

// Additions to Polyline.java

```
public class Polyline extends Arc{
    // ...

    public double computeInkCost() {
        double inkCost = 0;

        for (int i = 0; i < nodeList.size() - 1; i++) {
            inkCost += width*nodeList.get(i).distanceToNode(
                nodeList.get(i + 1));
        }
        return inkCost;
    }
}
```

// Additions to Circular.java

```
public class Circular extends Arc{
    // ...

    public double computeInkCost() {
        double radius = start.distanceToNode(center);
        return width * (radius * getRadians());
    }
}
```

ב. (3 נקודות)

השלם את מימוש ה copy constructor של המחלקה Segment, דהיינו בנאי המקבל עצם מטיפוס Segment ומשכפל אותו:

```
// copy constructor: creates a duplicate of the parameter
Segment(Segment anotherSegment) {
    super(anotherSegment.width, anotherSegment.color);
    this.start = anotherSegment.start;
    this.end = anotherSegment.end;
}
```

ג. (4 נקודות)

יכולנו להגדיר את Polyline כרשימה של עצמים מטיפוס Segment. מה היתרון של הגישה אותה נקטנו?

חיסכון במקום.

שדות הצבע והעובי לא משוכפלים עבור כל אחד מהמקטעים. כמו כן, בדרך זו כל קודקוד (פנימי) נשמר רק פעם אחת (ולא בשני ה Segments שבהם הוא חל).

ומה החסרון?

שכפול קוד.

הפונקציונאליות של Segment (נוכחית ועתידית!) תשוכפל ויהיה צורך לתחזק אותה גם עבור Polyline. ראה לדוגמא: toString.

ד. (10 נקודות)

ברצוננו לתמוך גם בהדפסה לתווין ישן יותר היודע לקבל קטעים ישרים (Segments) בלבד. ההמרה מתבצעת כך: קטע ישר נשאר אותו דבר, קו פוליגונאלי מתואר כאוסף הקטעים הישרים המרכיבים אותו, וקשת מעגלית מוחלפת בקטע הישר שמחבר את קצותיה.

ממשו את השרות `transformToSegmentList` המתרגם קשתות (Arcs) מכל הסוגים לקטעים ישרים (Segments). השרות מ קבל שני פרמטרים: `arcList` הוא רשימת קשתות המיועדת לתווין החדש ו- `segList` הוא פרמטר פלט שמקבל רשימת קטעים ריקה אשר תכיל בסיום השרות את תוצאת התרגום, דהיינו רשימת קטעים ישרים לתווין הישן.

בסעיף זה לא ניתן לשנות את מימושי המחלקות האחרות.

```
public class Plotter {

    // ...

    public static void transformToSegmentList(List<Arc> arcList,
                                             List<Segment> segList) {
        for (Arc arc: arcList) {
            if (arc instanceof Segment) {
                Segment s = (Segment) arc;
                Segment seg = new Segment(s); //copy constructor
                segList.add(seg);
            }

            if (arc instanceof Circular) {
                Circular circ = (Circular) arc;
                Segment seg = new Segment(circ.getWidth(),
                                           circ.getColor(),
                                           circ.getStartNode(),
                                           circ.getEndNode());

                segList.add(seg);
            }

            if (arc instanceof Polyline) {
                Polyline poly = (Polyline) arc;
                List<Node> nodeList = poly.getNodeList();
                for (int i=0; i < nodeList.size()-1; i++) {
                    Segment seg = new Segment(poly.getWidth(),
                                               poly.getColor(),
                                               nodeList.get(i),
                                               nodeList.get(i+1));

                    segList.add(seg);
                }
            }
        }
    }
}
```

ה. (5 נקודות)

כעת אנו מעוניינים לאפשר גמישות בהגדרת צומת (Node). המשתמש במחלקת שלנו יוכל לספק טיפוס צומת כרצונו (בכפוף לכללים נתונים כמובן). שכתבו את המחלקה Polyline כך שתהיה גנרית מבחינת הטיפוס Node.

```
public class Polyline<NODE_T> extends Arc {
    private List<NODE_T> nodeList;

    public Polyline(double width, Color color,
                    List<NODE_T> nodeList){
        super(width, color);
        this.nodeList = new ArrayList<NODE_T>(nodeList);
    }

    public List<NODE_T> getNodeList() {return nodeList;}
}
```


שאלה 2 (32 נקודות)

בשאלה זו נגדיר מחלקה המכילה מערך אשר אבריו קובעים איזו פעולה תתבצע.

א. (15 נקודות)

הגדירו מחלקה בשם ExecutionEngine ובה מערך בשם commands (טיפוס המערך יקבע על פי האופן בו אתם פותרים את השאלה). על המחלקה לתמוך בביצוע פקודות חסרות פרמטרים וללא ערך מוחזר (void).

הגדירו בתוך המחלקה ExecutionEngine:

1. שרות בשם setCommand אשר בעזרתו המשתמש מעביר פקודה לביצוע (אתם קובעים באיזה אופן) ומספר שלם, והפקודה "נשמרת" במערך במקום זה (המספר השלם משמש כאינדקס). שימו לב כי השרות אינו מבצע את הפקודה שקיבל כפרמטר.
2. שרות בשם exec המקבל מספר שלם ומבצע את הפקודה הרלוונטית לפי האיבר במערך המתאים למספר זה (אינדקס).

על המחלקה ExecutionEngine לספק 2 פקודות לדוגמא, כך שמשתמשי המחלקה יוכלו להשים פקודות אלו ישירות למערך (בעזרת השרות setCommand) ולא יצטרכו לממש אותן. הפקודות לדוגמא הן:

1. פקודה המדפיסה את לוח הכפל בגודל 10 על 10
2. פקודה אשר מבקשת מהמשתמש מחרוזת ומדפיסה אותה הפוכה. ב"מבקשת מהמשתמש מחרוזת" אנו מתכוונים להדפסת הודעה למסך, וקבלת קלט משתמש מהמקלדת.

שימו לב, הלקוחות של המחלקה ExecutionEngine יכולים להגדיר פקודות נוספות משל עצמם.

על התוכנית לא לעוף בזמן ריצתה בשום מקרה שאינו בלתי נמנע. דרך הטיפול בחריגים היא לשיקולכם, ונקבל כל תשובה סבירה. מותר להגדיר מבני עזר מחוץ למחלקה ExecutionEngine אם הדבר נדרש או מפשט את הקוד ובלבד שהמבנה של הקוד הוא הגיוני, ראוי ונכון.

כמו כן יש להגדיר למחלקה שרותים ו/או בנאים נוספים לפי הצורך.

```
public interface Command {
    public void run();
}
```

```
import java.util.Scanner;

public class ExecutionEngine {

    private Command [] commands;

    /**
     * @pre for every i=0..commands.length-1 commands[i] != null
     */
    public ExecutionEngine(Command[] commands){
        this.commands = commands;
    }

    /**
     * @pre c != null
     * @pre 0 <= index < numOfAvailableCommands()
     * @post exec(index) will execute c.run()
     */
    public void setCommand(Command c, int index){
        commands[index] = c;
    }

    /**
     * @pre 0 <= index < numOfAvailableCommands()
     */
    public void exec(int index){
        commands[index].run();
    }

    public int numOfAvailableCommands(){
        return commands.length;
    }

    public static Command printMultChart = new Command() {
        public void run(){
            for (int i = 1; i <10; i++) {
                for (int j = 1; j < 10; j++) {
                    System.out.print(i * j + "\t");
                }
                System.out.println();
            }
        }
    };

    public static Command stringReverse = new Command() {
        public void run(){
            System.out.println("Enter a String to reverse: ");
            Scanner s = new Scanner(System.in);
            String str = s.nextLine();

            for (int i = str.length()-1; i > -1; i--) {
                System.out.print(str.charAt(i));
            }
        }
    };
}
```

ב. (7 נקודות)

הגדירו את החוזה של השרות `setCommand` אותו מימשתם בסעיף א. על החוזה לתאר את תנאי הקדם והבתר של השרות תוך שימוש בטענות `Design by Contract` כנלמד. אם יש תנאים אותם אין ביכולתכם לבטא בתחביר פורמלי הוסיפו הערות מילוליות.

```
/**
 * @pre c != null
 * @pre 0 <= index < numOfAvailableCommands()
 * @post exec(index) will execute c.run()
 */
public void setCommand(Command c, int index){
```

הגדירו את החוזה של השרות `exec` אותו מימשתם בסעיף א. על החוזה לתאר את תנאי הקדם והבתר של השרות תוך שימוש בטענות `Design by Contract` כנלמד. אם יש תנאים אותם אין ביכולתכם לבטא בתחביר פורמלי הוסיפו הערות מילוליות.

```
/**
 * @pre 0 <= index < numOfAvailableCommands()
 */
public void exec(int index)
```

ג. (10 נקודות)

עליכם לממש לקוח של המחלקה `ExecutionEngine`.

הלקוח ייצור בשרות ה-`main` שלו מופע של המחלקה `ExecutionEngine` ויאתחל אותו עם הפקודות המובנות (להדפסת לוח הכפל והפיכת מחרוזת) וכן עם פקודה חדשה אשר מדפיסה את עשרת **המספרים המושלמים** הראשונים (ראו הגדרה בהמשך). לאחר מכן הלקוח יריץ בעזרת `ExecutionEngine` את 3 הפקודות.

מותר להגדיר במחלקת הלקוח שרותים נוספים וכן מבני עזר מחוץ למחלקת הלקוח אם הדבר נדרש או מפשט את הקוד.

מספר מושלם הוא מספר טבעי השווה לסכום כל המספרים הטבעיים הקטנים ממנו המחלקים אותו ללא שארית. המספר המושלם הראשון הוא 6 מכיוון שהוא מתחלק ב-1, 2 ו-3 ומתקיים $6 = 1+2+3$ ואחריו בא 28 $(1+2+4+7+14=28)$.

```
public class ExecutionEngineClient {

    public static void main(String[] args) {
        Command [] commands = new Command[3];
        commands[0] = ExecutionEngine.printMultChart;
        commands[1] = ExecutionEngine.stringReverse;
        commands[2] = perfectTen;

        ExecutionEngine engine = new ExecutionEngine(commands);
        engine.exec(0);
        engine.exec(1);
        engine.exec(2);
    }

    public static Command perfectTen = new Command() {
        public void run() {
            int i = 0;
            int candidate = 1;
            while(i < 10){
                if (isPerfect(candidate)){
                    System.out.println(candidate);
                    i++;
                }
                candidate++;
            }
        }
    };

    public static boolean isPerfect(int num){
        Set<Integer> devidors = devidors(num);
        int sumDevidors = 0;
        for (Integer integer : devidors) {
            sumDevidors += integer;
        }

        return sumDevidors == num;
    }

    public static Set<Integer> devidors(int num){
        Set<Integer> result = new TreeSet<Integer>();
        for (int i = 1; i < num; i++) {
            if(num % i == 0)
                result.add(i);
        }
        return result;
    }
}
```

שאלה 3 (20 נקודות)

במערכות תוכנה מורכבות אנו מעוניינים לשלוט על מספר המופעים שיווצרו מטיפוס מסוים. למשל, כאשר יש במערכת שעון (או מונה) המיוצג על ידי המחלקה Clock, איננו רוצים שיווצרו שני שעונים (אולי ע"י מודולים שונים אשר לא נכתבו ע"י אותו המתכנת), וכך המערכת תצא מסכרון, כי כל שעון עלול להחזיק ערך שונה.

תבנית העיצוב Singleton פותרת בעיה זו בדיוק. הקוד הבא אינו מאפשר יצירת יותר ממופע אחד של המחלקה Singleton1:

```
01 public class Singleton1 {
02
03     private static Singleton1 instance;
04
05     private Singleton1() {
06         // ...
07     }
08
09     public static Singleton1 getInstance(){
10         if (instance == null)
11             instance = new Singleton1();
12         return instance;
13     }
14
15     // more code...
16 }
```

בסעיפים הבאים נציג 3 מימושים חלופיים ל Singleton1. ואתם תתבקשו לדון בהבדלים שבין המימושים השונים.

א. (5 נקודות)

נתונה המחלקה Singleton2:

```
01 public class Singleton2 {
02
03     private static Singleton2 instance = new Singleton2();
04
05     private Singleton2() {
06         // ...
07     }
08
09     public static Singleton2 getInstance() {
10         return instance;
11     }
12     // more code...
13 }
```

1. הסבירו מה משמעות ההבדלים בין מימוש Singleton1 ו-Singleton2 ?

מ-Singleton1 יוצר מופע אחד לכל היותר, וזאת רק לאחר הקריאה הראשונה ל-
getInstance. מ-Singleton2 יוצר מופע אחד בדיוק בעת טעינת המחלקה לזכרון.

2. באילו מקרים נעדיף להשתמש ב-Singleton1 ?

אם לא בכל שימוש אריצה של התוכנה יהיה צורך ביצירת Singleton1.

3. באילו מקרים נעדיף להשתמש ב-Singleton2 ?

- אם יצירת המופע דורשת משאבים (זמן או זכרון) שאין ביכולתנו להקצות כאשר המערכת רצה, נעדיף להקצותם בהתחלה
- אם יש משמעות ליצירת המופע גם ללא שימוש בו (מחלקה ש"רצה ברקע")

ב. (5 נקודות)

נתונה המחלקה Singleton3:

```
01 public class Singleton3 {
02
03     private static Singleton3 instance;
04
05     Singleton3() {
06         // ...
07     }
08
09     public static Singleton3 getInstance(){
10         if (instance == null)
11             instance = new Singleton3();
12         return instance;
13     }
14     // more code...
15 }
```

1. הסבירו מה משמעות ההבדלים בין מימוש Singleton1 ו-Singleton3 (שורה 5)?

מחלקות בחבילה של Singleton3 יכולות ליצור יותר ממופע אחד של המחלקה

2. ספקו דוגמת קוד הממחישה את ההבדלים בין מימוש Singleton1 ו-Singleton3

הקוד: Singleton3 s = new Singleton3();
יתקמפל רק מתוך החבילה שבה נמצאת Singleton3

הקוד: Singleton1 s = new Singleton1();
לא מתקמפל כלל

3. באילו מקרים נעדיף להשתמש ב-Singleton3?

במקרים שבהם אנו סומכים על מחלקות מאותה חבילה שייצרו יותר ממופע אחד וידעו לתאם בין המופעים השונים

ג. (5 נקודות)
נתונה המחלקה Singleton4:

```
01 public class Singleton4 {
02
03     private static Singleton4 instance;
04
05     private Singleton4() {
06         // ...
07     }
08
09     public Singleton4 getInstance() {
10         if (instance == null)
11             instance = new Singleton4();
12         return instance;
13     }
14     // more code...
15 }
```

1. הסבירו מה משמעות ההבדלים בין מימוש Singleton1 ו-Singleton4 (שורה 9)?

מהמחלקה Singleton4 לא ניתן לייצר מופעים כלל

2. איזה קטע קוד שאינו שרות מחלקה (מתודה סטטית) צריך להופיע במחלקה (למשל בשורה 14) כך שניתן יהיה ליצור מופעים ממנה?

למשל, **בנאי מועמס:**

```
public Singleton4(int dummy){}
```


ד. (5 נקודות)

נתון קטע הקוד הבא:

```
01 class SubSingleton1 extends Singleton1 {
02
03     private static SubSingleton1 instance;
04
05     private SubSingleton1() {}
06
07     public static SubSingleton1 getInstance(){
08         if (instance == null)
09             instance = new SubSingleton1();
10         return instance;
11     }
12
13 }
14
15 public class SomeClient{
16
17     public static void main(String[] args) {
18         SubSingleton1 s1 = SubSingleton1.getInstance();
19         SubSingleton1 s2 = SubSingleton1.getInstance();
20         System.out.println("s1 == s2 is " + (s1==s2));
21     }
22 }
```

האם הקוד יגרום ל:

שגיאת קומפילציה (אם כן, באיזו שורה ומה הסיבה) / תעופה בזמן ריצה (אם כן, באיזו שורה ואיזה חריג יזרק) / הקוד תקין (אם כן, מה יודפס?)

בשורה 5 יש קריאה מרומזת לבנאי הפרטי של Singleton1

שאלה 4 (16 נקודות)

נתון הממשק הבא:

```
public interface I {
    public void m(String s);
}
```

לכל אחת מהמחלקות והממשקים הבאים ציינו אם הוא חוקי או שתהיה שגיאת קומפילציה, ואז יש לציין מהי השגיאה.

```
public abstract class C implements I {
    public abstract void m(String s){ };
}
```

א. חוקי / שגיאת קומפילציה – מהי? למתודה מופשטת אסור שיהיה גוף

```
public abstract class C implements I {
}
```

ב. חוקי / שגיאת קומפילציה – מהי?

```
public class C extends I {
    public void m(int i){ };
}
```

ג. חוקי / שגיאת קומפילציה – מהי? המילה extends שגויה, צריך implements

```
public class C implements I {
    public void m(int i){ };
}
```

ד. חוקי / שגיאת קומפילציה – מהי? לא ממש את m(String)

```
public class C implements I {
    public void m(String s){ };
    public void m(int i){ };
}
```

ה. חוקי / שגיאת קומפילציה – מהי?

```
public interface J extends I {
}
```

ו. חוקי / שגיאת קומפילציה – מהי?

```
public interface J extends I {
    public void m(int i) throws Exception;
}
```

ז. חוקי / שגיאת קומפילציה – מהי?

```
public interface J extends I {
    public void m(String i) throws Exception;
}
```

ח. חוקי / שגיאת קומפילציה – מהי? פסוק throws לא תואם את מה שהוגדר במתודה הנדרסת

בהצלחה!!

נספח: חלק מתיעוד הממשק List

java.util **Interface List<E>**

Method Summary

boolean	add (E o) Appends the specified element to the end of this list (optional operation).
void	add (int index, E element) Inserts the specified element at the specified position in this list (optional operation).
E	get (int index) Returns the element at the specified position in this list.
boolean	isEmpty () Returns true if this list contains no elements.
Iterator < E >	iterator () Returns an iterator over the elements in this list in proper sequence.
E	remove (int index) Removes the element at the specified position in this list (optional operation).
boolean	remove (Object o) Removes the first occurrence in this list of the specified element (optional operation).
boolean	removeAll (Collection <?> c) Removes from this list all the elements that are contained in the specified collection (optional operation).
E	set (int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).
int	size () Returns the number of elements in this list.
Object []	toArray () Returns an array containing all of the elements in this list in proper sequence.
<T> T[]	toArray (T[] a) Returns an array containing all of the elements in this list in proper sequence; the runtime type of the returned array is that of the specified array.