מבחן ב- "תכנות מונחה עצמים" + הצעת פתרון

סמסטר א' 2020, מועד ב', 1.03.2020, מספר קורס:7029910 - 1-5
פרופ' בעז בן משה, גב' אליזביט איצקוביץ

משך הבחינה שעתיים וחצי
חומר עזר: אין (אסור ספרים, אסור מחברות אסור מחשבים)

במבחן זה 4 שאלות, יש לענות על כולן.

תשובות מסורבלות או ארוכות מדי לא יזכו בניקוד מלא. הקפידו על כתב יד ברור והסברים (ניתן בהחלט לתעד בעברית).

מותר לפתור סעיף מסוים ע"י שימוש בסעיפים אחרים

מבנה הבחינה:
עמוד 1: דף הנחיות כללי.
עמוד 2: שאלות הבחינה
עמודים 3-4: חומר עזר לבחינה

הקפידו על טוהר הבחינה!!

בהצלחה!

שאלה 1 (25 נקודות):

התייחסו לממשק של graph שמייצג גרף מכוון ממושקל (מצורף מטה). נגדיר שגרף g1 **מכיל** את גרף g2 אם ורק אם קבוצת הקדקודים של g1 מכילה את קבוצת הקדקודים של g2, ולכל צלע שנמצאת ב g2 קיימת צלע ב g1ששווה לה.

הדרכה: ניתן להניח ששני קדקודים שווים אם ורק אם יש להם אותו ערך getKey(), ושתי צלעות שוות אם ורק אם יש להן אותו קודקוד מקור, אותו קודקוד יעד, ואותו משקל.

1.1 (5 נקודות) הסבירו (במילים ובקיצור) כיצד נכון לממש פונקציה שבודקת האם גרף מכוון אחד מכיל גרף אחר.

1.2 (15 נקודות) כתבו פונקציה סטטית שמקבלת שני גרפים ומחזירה אמת אם ורק אם g1 מכיל את הגרף g2.
static boolean contains(graph g1, graph g2);

1.3 (5 נקודות) כתבו פונקציה סטטית שמקבלת שני גרפים ומחזירה:

- 0: אם הם שווים
- 1: אם g1 מכיל (ממש) את g2.
- 2: אם g2 מכיל (ממש) את g1.
- 1-: אחרת.

static int relation(graph g1, graph g2)

תשובה:

1.1 ניקח את אוסף כל הקדקודים של כל אחד מהגרפים (v1,v2) בהתאמה. נעבור על כל קודקוד ב v2 אם הוא אינו קיים ב v1 - נחזיר false, בהנחה שקיים, לכל קודקוד ב v2 ניקח את רשימת הצלעות שיוצאות ממנו ועבור כל צלע נבדוק שהיא אכן קיימת ב g1.

1.2

```
public static final int NONE=-1, EQAUL = 0, C12 = 1, C21=2;
static boolean contains(graph g1, graph g2) {
	boolean ans = true;
	Collection<node_data> v1 = g1.getV();
	Collection<node_data> v2 = g2.getV();
	Iterator<node_data> i2 = v2.iterator();
	while(ans & i2.hasNext()) {
		node_data c2 = i2.next();
		if(!v1.contains(c2)) {return false;}
		Collection<edge_data> e2 = g2.getE(c2.getKey());
		Iterator<edge_data> ei2 = e2.iterator();
		while(ans && ei2.hasNext()) {
			edge_data ce2 = ei2.next();
			edge_data ce1 = g1.getEdge(ce2.getSrc(), ce2.getDest());
				if(!ce2.equals(ce1)) {ans = false;}
		}
	}
	return ans;
}

static int relation(graph g1, graph g2) {
	boolean c12 = contains(g1, g2);
	boolean c21 = contains(g2, g1);
	if(c12 & c21) return EQAUL;
	if(c12) return C12;
	if(c21) return C21;
	return NONE;
}
```

שאלה 2 (25 נקודות):

הניחו שקיימת לכם מחלקה בשם DGraph שמממשת את הממשק graph (מצורף לבחינה, ראו מטה):

2.1 (12 נקודות) הסבירו כיצד ניתן לממש את המחלקה DGraph_Undo שמממשת את המשק graph אבל יש לה גם שיטה

public void undo();

שמחזירה את הגרף להיות במצב הקודם שלו – שימו לב שכמות הפעמים שניתן לעשות undo אינה מוגבלת, אם אין יותר מה לעשות הגרף צריך להישאר במצבו הראשוני.

2.2 (13 נקודות) ממשו את המחלקה DGraph_Undo

תשובה:

2.1 כללית נבנה את המחלקה DGraph_Undo שתירש מ DGraph. נוסיף למחלקה החדשה במבנה נתונים של "מחסנית" (ניתנת למימוש פשוט למשל ע"י ArrayList<graph>), אשר לפני כל שינוי בגרף נשמור במקום העליון (הראשון – אינדקס 0) במחסנית העתק (עמוק) של הגרף הנוכחי ואז נבצע את שינוי בגרף (ע"י קריאה לשיטה מהמחלקה DGraph – ממנה ירשנו). פעולת ה undo תשלוף את האיבר הראשון (אם קיים) במחסנית ותעדכן את הגרף הנוכחי בעזרת השיטה של init.

2.2

```
public class DGraph_Undo extends DGraph {
	private ArrayList<graph> _undo_stack;
	public DGraph_Undo() {  // empty (default) constructor
		super();
		_undo_stack = new ArrayList<graph>(); //init the "Stack".
	}
	public void addNode(node_data n) {
		update_undo_stack();  // save a "snapshoot" of this graph
		super.addNode(n);     // update the graph
	}
	public void connect(int src, int dest, double w) {
		update_undo_stack();
		super.connect(src, dest, w);}
	public node_data removeNode(int key) {
		update_undo_stack();
		return super.removeNode(key);}
	public edge_data removeEdge(int src, int dest) {
		update_undo_stack();
		return super.removeEdge(src, dest);}
	public void undo() {
		if(_undo_stack.size()>0) {
		graph last = _undo_stack.remove(0); //removes the last
			this.init(last); // update this graph with it
		}
	}
	/////// PRIVATE METHODS FOR UNDO IMPLEMNTATION ////////
	private void update_undo_stack() {
		graph cc = this.copy();  // deep copy
		_undo_stack.add(0, cc);  // stores the current graph
	}
}
```

שאלה 3 (25 נקודות):

ענו על השאלות הבאות – שימו לב בשאלה זאת אין צורך לכתוב קוד:

3.1 (12 נקודות) הסבירו מהו S.O.L.I.D. בהקשר של תבניות עיצוב.

3.2 (13 נקודות) נניח שקיימות שתי שיטות שונות לביצוע **אותו** חישוב מסובך כלשהו:

long m1(List<String> a);

long m2(List<String> a);

הסבירו כיצד ניתן לכתוב שיטה אחת:

long m12(List<String> a)

שמאפשרת להריץ את שתי השיטות (m1,m2) במקביל ולהחזיר את הערך הראשון שמחושב ע"י השיטה הראשונה שמסתיימת.

הדרכה – אין צורך לכתוב קוד, ניתן להניח שהקלט לשיטות אינו null ואינו ריק.

תשובה:

3.1: תשובה ממוקדת ניתן למצוא בדף ה wiki של solid,

צירוף האותיות SOLID- הוא ראשי תיבות של:

Single Responsibility Principle (SRP)

Open Closed Principle (OCP)

Liskov Substitution Principle (LSP)

Interface Segregation Principle (ISP)

Dependency Inversion Principle (DIP)

מפ"מ = מחלקות, פונקציות, מודולים.

עיקרון ה Single Responsibility Principle-מדבר על תחום האחריות של המפ"מ , לכל מפ"מ יש אחריות אחת בלבד

עיקרון ה Open Closed Principle-מדבר על כך שהמפ"מ יהיה פתוח להרחבות (Extensions) אך סגור לשינויים(Modifications)

עיקרון ה) Liskov Substitution Principle-על שם ברברה ליסקוב שזכתה בפרס טיורינג הנחשב ב-2008) מדבר על כך שאנו יכולים להחליף אובייקטים של מחלקות שונות אשר יש ביניהם קשר של ירושה, ניתן להחליף אובייקט של מחלקת אבא באובייקט של מחלקת בן וזה מבלי לשנות את ההתנהגות של האב.

עיקרון ה Interface Segregation Principle-מדבר על אופן עבודה בו אנו נרצה להפריד בין ממשקים שונים בכדיי למנוע מימוש של תהליכם לא נחוצים, במילים אחרות – לא נתקמצן ביצירת ממשקים (כשזה נחוץ כמובן)

עיקרון ה Dependency Inversion Principle-מדבר על ההפרדה בין הקומפוננטות למען תחזוקה טובה יותר של הקוד וזה מתבצע על ידי אבסטרקציה תוך שימוש ב Intefraces-במקום מחלקות

3.2 פתרון פשטני יתבסס על ייצור של שלושה תהליכים: תהליך מרכזי של הפונקציה m12, שיפעיל שני תהליכים של m1,m2 ויבצע wait על הפרמטר המשותף (List<String> a), כל אחד מהתהליכים של m1 וm2 יריצו את הפונקציה שלהם וכאשר כל אחד מהם יסתיים הוא יעדכן שדה משותף ויבצע notify. הערך המוחזר יהיה הערך של השדה המשותף.

<u>שאלה 4 (25 נקודות):</u>
בשאלה זו עליכם לתכנן מערכת תוכנה – אין צורך לכתוב קוד.
נסתכל על מערכת chat שמאפשרת ללקוחות שונים להתחבר לשרת ואז לקבל את רשימת המשתמשים, לאחר מכן כל לקוח יכול לבחור האם לשלוח הודעה ללקוח אחר או לכולם, כאשר ברגע שלקוח מסוים מקבל הודעה (אישית או קבוצתית) היא צריכה להיות מוצגת.
4.1 (13 נקודות) ציינו את המחלקות העיקריות במערכת, לכל מחלקה ציינו את השיטות המרכזיות ושדות המידע העיקריים, שרטטו דיאגרמת מחלקות.
4.2 (12 נקודות) הסבירו את התהליכים במערכת, עמדו על אופן הפעולה שלהם כולל התייחסות לנושאים של thread safe.

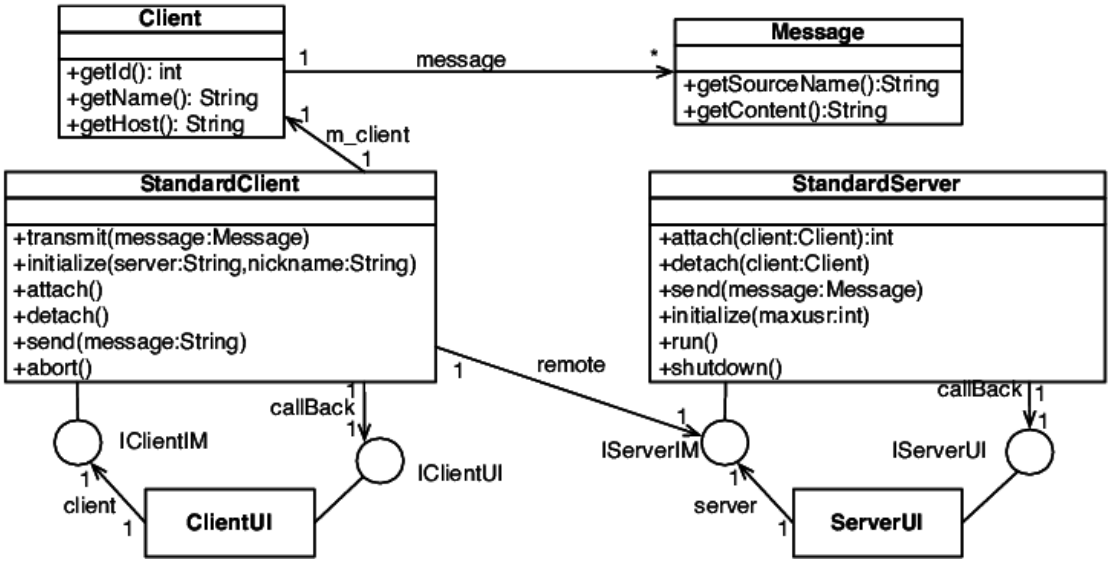
<u>תשובה:</u>

4.1 בפתרון נעשה שימוש במחלקות הבאות:

- שרת: המחלקה תכלול מבנה נתונים שיכיל מידע על כל אחד מהלקוחות. השרת יכלול שיטות (מסונכרנות – על בסיס הנתונים) להוספת משתמש, לחיבור משתמש, לניתוק משתמש, לפרסום כלל המשתמשים וכן שיטה לשליחת הודעה ממשתמש ליעד אחר (משתמש או כולם). השרת יכלול תהליך לתקשורת עם כל אחד מהלקוחות.

- לקוח: מחלקה שמייצגת את צד השרת, תכלול תהליך לטיפול ביצירת הודעה, ותהליך להקשבה להודעות.
- מחלקה לתצוגה של מידע לקוח: המידע יכול להיות מוצג טקסטואלית או בעזרת מחלקת עזר להצגה של ממשג גרפי. המחלקה תכלול שיטות להצגת הודעות, חיבור וניתוק מהשרת (וכן רישום אליו), וכן שיטה ליצירת הודעה.
- מחלקה הודעה: מייצגת הודעת טקסט, עם מקור, יעד (או יעדים) גוף הודעה וזמן.

להלן דוגמא סטנדרטית לדיאגרמת מחלקות מעלה:



דוגמא סטנדרטית לדיאגרמת מחלקות (מקור), דוגמא נוספת ניתן למצוא באתר הקורס - שימו לב שלצורך הפשטות נעשה שימוש במחלקות פנימיות – לא נדרש בשאלה זו.

4.2 כפי שנכתב מעלה: השרת יכיל תהליך לכל לקוח. משמע עבור כל לקוח שיתחבר לשרת יאותחל בשרת תהליך שיטפל התקשורת למול הלקוח. השרת יחזיק מבנה נתונים מסונכרן של לקוחות רשומים (ופעילים). מבחינת הלקוח הוא יכיל שני תהליכים: אחד להקשבה להודעות השרת והשני למשלוח הודעות (פרטיות או כלליות). כדי לוודא שהמערכת תקינה ועובדת בתצורה של thread safe, נוודא שבפונקציונליות כמו הוספת משתמש לשרת תתבצע כפונקציה מסונכרנת (על אוסף המשתמשים) כדי למנוע סיטואציה ששני לקוחות בעלי אותו שם יתחברו לשרת. גם בצד של שליחת הודעה ללקוח נרצה ולוודא שהודעה מהשרת ללקוח נשלחת באופן נשלחה במלואה לפני שנשלחת הודעה נוספת – ניתן לממש זאת למשל ע"י תור של הודעות בתהליך שמטפל במשלוח הודעות ללקוח.

חומר עזר לבחינה:

- ממשק שמייצג גרף מכוון ממושקל: graph,
- ממשק שמייצג קודקוד בגרף: node_data, ממשק שמייצג צלע בגרף: edge_data.
- נקודה דו מימדית Point2D

```
/** This interface represents a directional weighted graph. */
public interface graph {
	/** return a deep copy of this graph.*/
	public graph copy();
	/** init this graph to be a deep copy of g.*/
	public void init(graph g);

	/**return the node_data by the node_id (null - if none)*/
	public node_data getNode(int key);
	/** return the data of edge(src,dest), null if none */
	public edge_data getEdge(int src, int dest);
	/** add a new node to the graph with the node_data. */
	public void addNode(node_data n);
/** Connects an edge with weight w between src->dest. */
	public void connect(int src, int dest, double w);
	/** Return a reference (shallow copy) for the
	 * collection representing all the nodes in the graph. */
	public Collection<node_data> getV();
	/**
	 * Return a reference (shallow copy) for the
	 * collection representing all the edges getting out of
	 * the given node (src)
	 */
	public Collection<edge_data> getE(int src);
	/** Delete the node (with the given ID) from the graph */
	public node_data removeNode(int key);
	/** Delete the edge from the graph */
	public edge_data removeEdge(int src, int dest);
	/** return the number of nodes in the graph. */
	public int nodeSize();
	 * return the number of edges (directional graph). */
	public int edgeSize();
/** return the Mode Count for testing changes in the graph. */
	public int getMC();
}
/**
 * This interface represents the set of operations applicable on a
node (vertex) in a (directional) weighted graph. */
public interface node_data {
	/** Return the key associated with this node. */
	public int getKey();
}

/** This interface represents the set of operations applicable on a
directional edge(src,dest) */
public interface edge_data {
	/** @return the id of the source node of this edge. */
	public int getSrc();
	/** @return the id of the destination of this edge. */
	public int getDest();
	/** @return the weight of this edge (positive value). */
	public double getWeight();
}
```

```java
/** This class represents a 2D point in the plane. */
public class Point2D {
    public static final double EPS = 0.00001;
    public static final Point2D ORIGIN = new Point2D(0,0);
    private double _x,_y;
    public Point2D(double x,double y) {_x=x; _y=y;}
    public Point2D(Point2D p) {this(p.x(), p.y());}
    public double x() {return _x;}
    public double y() {return _y;}
    public Point2D add(Point2D p) {
      Point2D a = new Point2D(p.x()+x(),p.y()+y());
      return a;
    }

    public String toString() {return _x+","+_y;}

    public double distance() {return this.distance(ORIGIN);}
    public double distance(Point2D p2) {
        double dx = this.x() - p2.x();
        double dy = this.y() - p2.y();
        double t = (dx*dx+dy*dy);
        return Math.sqrt(t);
    }

    public boolean equals(Object p)
    {
        if(p==null || !(p instanceof Point2D)) {return false;}
        Point2D p2 = (Point2D)p;
        return ( (_x==p2._x) && (_y==p2._y));
    }
    public boolean close2equals(Point2D p2)
    {
        return ( this.distance(p2) < EPS );
    }
}
//////////////////////////////////////////////////////
```