מבחן ב- "מבוא לתכנות מונחה עצמים" + הצעת פתרון

סמסטר א' 2019, מועד א', 29.1.2019, מספר קורס: 7027910
אליזביט איצקוביץ, בעז בן משה

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 2-3 שאלות הבחינה

הקפידו על טוהר הבחינה!!

בהצלחה!

שאלה 1 (25 נקודות)

בשאלה זו נממש מנגנון "קודן לרכב", מנגנון זה מאפשר למשתמש להכניס קוד של 4 ספרות (בדוגמא זו נניח שישנן בסך הכל 6 כפתורים #,0,1,2,3,4 כאשר הקוד מורכב מ הספרות 0-4 כולל חזרות ובסופו מקישים #). אם הקוד שהוכנס נכון הרכב ניתן ל"הנעה" למשך דקה, וכל עוד הוא מונע המנגנון נשאר "פתוח". אם הקוד שהוכנס אינו נכון הרכב נשאר במצב "לא ניתן להנעה" ואם מקישים קוד שגוי 3 פעמים הקודן ננעל ל 15 דקות.

1.1 (15 נקודות) ממשו את המחלקה "קודן" (MyCode) שמממשת את הממשק car_code:
1.2 (10 נקודות) כתבו מחלקה בשם MyCodeTest (Junit) לבדיקת הפונקציונליות של המחלקה MyCode

```
public class MyCode implements car_code {
        // …
        public car getCar() {;} // הניחו ששיטה זו קיימת – אין צורך לממש
}

public interface car_code{
        public boolean open(String key); //  מאפשרת לפתוח את הקודן לפי החוקים
        public boolean isOpen(); // מחזירה את הסטטוס של הקודן (ניתן פתוח או סגור)
        public boolean canBeOpen();// מחזירה אמת אא"ם ניתן לפתוח כעת את הקודן.
        public car getCar(); // הניחו שהמימוש קיים - אין צורך לממש שיטה זו!
}
public interface car {
        public boolean isRunning(); //  מחזירה אמת אם ורק אם הרכב מונע.
        public boolean open(String key);// מאפשרת פתיחת הרכב, מחזירה סטטוס
        public Date lastRun(); // מחזירה את המועד האחרון שבו הרכב היה מונע
}
```

הדרכה:

- תזכורת: המחלקה Date מייצגת זמן, ויש לה בנאי ריק שבונה את הזמן הנוכחי, וכן יש לה שיטה בשם long getTime() שמחזירה את הזמן שעבר מה 1.1.1970 באלפיות השנייה.
- לצורך מימוש הבדיקות ניתן להניח שהקוד לרכב הוא "#1231"

הצעת פתרון שאלה 1:

1.1 ככלל ניתן היה לענות על שאלה זו במספר דרכים, הפתרון המוצג כאן אינו כולל שימוש בתהליכים (ניתן לעשות זאת אם כי בהחלט לא נדרש) – נדרש לעשות שימוש בשיטה getCar הן לצורך הפתיחה על הקוד והן לצורך הבדיקה האם הרכב מונע.
מגוון גדול של פתרונות התקבלו כנכונים גם במקרים של שגיאות ניסוח קלות ושל בדיקה חלקית של המקרים.

```
public class My_code implements car_code{
        public static final long TIME_OUT = 1000*60*15;
        public static final long MAX_TIME = 1000*60;
        public static final int MAX_ERR = 3;
        private car _car; // a reference for the actual Car object
        private long _lastERR, _lastOpen;
        private int _errNum; // counter for the number of wrong codes

        public My_code() {
                _car = new MyCar();
```

```
		_errNum = 0;
		_lastERR = 0; // aka none
		_lastOpen = 0; // not open
	}
	public boolean open(String key) {
		boolean ans = false;
		if(isOpen()) {ans= true;}
		if(!ans && canBeOpen()) {
			if(getCar().open(key)) {
				ans=true;
				_errNum=0;
				_lastOpen = now();
			}
			else {
				this._errNum ++;
				if(_errNum>=MAX_ERR) {
					this._lastERR = now();
				}
			}
		}
		return ans;
	}
	public boolean isOpen() {
		if(getCar().isRunning()) {return true;}
		if(now() - _lastOpen < MAX_TIME) {return true;}
		return false;
	}
	public boolean canBeOpen() {
		boolean ans = false;
		if(isOpen()) {ans= true;}
		if(!ans && now()-this._lastERR > TIME_OUT) {
			ans=true;
		}
		return ans;
	}
	public car getCar() {
		// No Need to Implement
		return _car;
	}
	private long now() {return new Date().getTime();}
}
```

1.2

```
class My_codeTest {
	public My_code my_code = new My_code();
	String right_code = "1231#";
	String[] wrong_codes = {"3231#", "dwd", "#####"};
	//@Before : one can possibly write the needed code in a “setup”
	@Test
	void testMy_code() {
		my_code = new My_code();
		boolean open = my_code.isOpen();
		assertFalse(open, "On init the car should not be open");
	}
	@Test
	void testOpen() {
		my_code = new My_code();
		boolean open = my_code.open(right_code);
```

```
            assertTrue(open,"Err: should be open - with right
code:"+right_code);
        }
        @Test
        void testOpenErr_OK() {
            my_code = new My_code();
            boolean open = my_code.open(wrong_codes[0]);
            assertFalse(open,"Err: should NOT be open - wrong code:");
            open = my_code.open(right_code);
            assertTrue(open,"Err: should be open:right code:"+right_code);
        }
        @Test
        void testCanBeOpen() {
            my_code = new My_code();
            for(int i=0;i<wrong_codes.length;i++) {
                boolean open = my_code.open(wrong_codes[i]);
                assertFalse(open,"Err: should NOT open: wrong code:");
            }
            boolean canBeOpen = my_code.canBeOpen();
            assertFalse(canBeOpen,"Err: should NOT be open - 3 errors!");
            boolean open = my_code.open(right_code);
            assertFalse(open,"Err: should NOT be opened - 3 errors");
        }
}
```

שאלה 2 (25 נקודות)

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

2.1 (8 נקודות) הסבירו מהו ממשק JDBC, הסבירו אילו שימושים נעשה בו בהקשר של מטלה 4 (המטלה האחרונה בקורס)

2.2 (8 נקודות) הסבירו באופן כללי היא מערכת github: הסבירו הפונקציות המרכזיות במערכת.

2.3 (9 נקודות) ציינו 4 תבניות עיצוב מרכזיות: עבר כל תבנית הסבירו בקצרה מה תפקידה ובאילו מקרים נכון להשתמש בה.

הצעת פתרון לשאלה 2:

2.1 ממשק JDBC הוא חלק מהספריות של השפה java, ומייצג גישה גנארית לבסיסי נתונים, ה API של JDBC מאפשר לבצע מספר פעולות מרכזיות בניהן:
טעינת driver (שנדרש להיות תואם לבסיס הנתונים שנרצה להשתמש בו). יצירת קישור Connection – שלמעשה יוצר ערוץ תקשורת עם בסיס הנתונים דרך הממשק.
יצירת פקודה (Statement) שלמעשה מהווה פקודת SQL, אותה ניתן להריץ ולקבל תוצאה בדמות תת טבלה – ResultSet, אוסף זה ניתן למעבר בצורה סדרתית.
היתרון בממשק הוא היכולת של מעבר בין בסיסי נתונים שונים כמעט ללא צורך לעשות שינויים בקוד.
במטלה 4 נעשה שימוש פשטני בבסיס נתונים: לאחר כל משחק תוצאת המשחק מבחינת, מזהי השחקנים, הזמן, והציון נשלחו לטבלה גלובלית בשרת כלשהו – עדכון זה נעשה למעשה בצורה אוטומטית ולא דרש את התערבות הסטודנטים. בהמשך נדרשו הסטודנטים לפתח אפליקציה בסיסית שניגשת לטבלה הגלובאלית דרך הממשק של JDBC ולקבל את התוצאות ולהשוות בין התוצאות של הקבוצה לתוצאות של קבוצות אחרות ביחס לתרחישים השונים שנבדקו (כ 9 במספר).

2.3+2.2 ראו הרצאה + תרגול בנושא באתר הקורס – חשוב לציין אנו נמנעים מלציין תשובה ספציפית כדי להקטין את "הפיתוי" ללמוד את התשובות בעלפה – חשוב להבין את המושגים והתחום באופן כללי

שאלה 3 (25 נקודות):

בשאלה זו נתייחס למימוש סימולציה של מיקום לאורך מסלול – כתלות בזמן.

נניח שיש לנו מסלול path, שלוקח 50 שניות (ונבנה מקובץ). סטודנט הציעה לממש את מערכת הסימולציה באופן הבא (הניחו תקינות, וקיום של מחלקה גרפית פשוטה בשם MyFrame):

```
public static void main(String[] args) {
        long dt = 100;// delta time in mili seconds
        long time = 0;// time from start
        MyFrame window = new MyFrame().show(); // GUI
        double MAX_TIME = 50*1000; // 50 seconds in mili seconds.
        Path path = new Path("path.txt");
        long last_time  = new Date().getTime();
        Point p = path.getPointInTime(time);
        while(time <MAX_TIME) {
                long now = new Date().getTime();
                if(now>=last_time + dt) {
                        time = time + (now-last_time);
                        last_time = now;
                        Point p = path.getPointInTime(time);
                        window.update(p);
                        window.repaint();
                } // if
        } // while
} // main
```

3.1 (8 נקודות) הסבירו באופן כללי מה לא נכון בלממש סימולציה בזמן באופן שמוצג מעלה.

3.2 (8 נקודות) הסבירו בפירוט כיצד נכון לממש מערכת של סימולציה בזמן (בסעיף זה אין צורך לכתוב קוד אלא רק להסביר).

3.3 (9 נקודות) ממשו את המערכת כך שתממש את אותה סימולציה אבל באופן נכון יותר.

הצעת פתרון – שאלה 3

3.1 הנקודה מתעדכנת כל dt אלפיות שנייה (עשירית שנייה) – הקוד נכון אבל הוא מבצע בדיקה רציפה (כל הזמן) האם כבר עברה עשירית שנייה מהעדכון הקודם – הריצה הזאת בתוך לולאת ה while גוזלת משאבים רבים. במילים אחרות אם אנחנו יודעים שצריך להמתין עשירית שנייה - אין צורך למדוד זמן בצורה רציפה ע"י מדידת זמן נוכחי – זה גורם שוב ושוב להפעלת while.

3.2 במקום להשתמש כל פעם בזמן הנוכחי יש להמתין dt שניות בעזרת פונקציה THREAD.sleep(dt) ולעדכן את הנקודה. כאן אין צורך ב- if. נעשה זאת בעזרת תהליך שנכתוב בעצמינו, נדגים שגם פתרונות שעשו שימוש ב Timers אפשריים ומקובלים.

3.3

```
class PointMoving extends Thread{
      long dt = 100;
      long time = 0;
      double MAX_TIME = 50*1000;
      MyFrame window;
      Path path;
      public Path(MyFrame window, Path path){
            this.window = window;
            this. path = path;
      }
      public void run() {
            while(time<MAX_TIME) {
                  Point p = path.getPointInTime(time);
                  window.update(p);
                  window.repaint();
                  try {
                        Thread.sleep(dt);
                  } catch (InterruptedException e) {}
                  time = time + dt;
            }//end while
      }//end run
}// PointMoving

public static void main(String[] args) {
      MyFrame window = new MyFrame().show(); //GUI
      Path path = new Path("path.txt");
      PointMoving p = new PointMoving(window, path);
      p.start();
      p.join();// not necessary
}// end main
```

שאלה 4 (25 נקודות):

במטלה מספר 4 עסקנו בתכנון תנועה בסביבה עם מכשולים, בשאלה זו נפתח מערכת שמייצגת אוסף מכשולים מלבניים דו ממדיים (המלבנים מקבילים לצירים) ומאפשרת יכולת לבדוק האם קטע מסוים זר למכשולים (לא נחתך ע"י המכשולים).

שימו לב: בשאלה זו אין צורך לכתוב קוד.

4.1 (7 נקודות) הסבירו באופן כללי כיצד עובדת המערכת.
4.2 (11 נקודות) ציינו והסבירו אילו מחלקות עליכם לממש – עבור כל מחלקה ציינו מה היא מייצגת, אילו שיטות ושדות מידע יש לה.
4.3 (7 נקודות) הסבירו בשרטוט ואו במילים אילו בדיקות צריך לעשות כדי לוודא שהמערכת עובדת (מבחינת בדיקות Junit).

הצעת פתרון שאלה 4

4.1 ככלל נתמקד בשאלה הבאה: בהינתן אוסף R של מלבנים (מקבילים לראשית הצירים) וקטע S (שמוגדר ע"י נקודות) האם S לא נחתך ע"י המלבנים משמע הוא זר לכל המלבנים.

- נשים לב שאם אחד מהקודים של S מוכל באחד המלבנים התשובה היא שלישית (הוא אינו זר) – ולכן ישר ניתן להחזיר False. כדי לחשב זאת נצטרך לכתוב פונקציה שמקבל נקודה ומלבן מקביל לראשית הצירים ומחשבת האם היא מוכלת בו.
- נשים לב שאם S נחתך ע"י אחד המלבנים לפחות אבל אף אחת מנקודות הקצה שלו אינן מוכלות ב R, אזי קיימת צלע אחת לפחות מתוך כל הצלעות של המלבנים ב R (למעשה שתיים) שחותכות את S. כדי לממש את הבדיקה הזאת נצטרך לכתוב פונקציה שמקבל סגמנט וקטע אנכי או אופקי ומחשבת האם יש להם נקודות חיתוך (למעשה ניתן להתחכם כאן ולבדוק רק מול שני אלכסוני המלבן – אבל זה אינו נדרש בפתרון זה – סתם נחמד).
  ניתן לממש פונקציה כזו באופן פשוט יחסי ע"י חלוקה לשני מקרים אופקי ואנכי ולהתייחס לסגמנט כפונקציה לינארית ולהציב בה את נקודות ה X במקרה של ישר אנכי או את נקודת ה Y במקרה של סגמנט אופקי (על הפונקנציה ההפוכה מ Y ל X). בהצבה נוכל לבדוק העם הנקודה נופלת בתוך הסגמנט לפי מדידת סכום המרחקים לשתי הנקודות שמגדירות האת הסגמנט.
- בעזרת שתי השיטות מעלה ניתן לבדוק האם הקטע זר לכל אוסף המלבנים – אם כן להחזיר True ואחרת False

4.2
מצורפות שתי תשובות אחת בעברית, והשנייה בטקסט באנגלית, גם תשובות בדיאגרמת מחלקות הוא פתרון אפשרי – בכל מקרה מספיקה גרסה אחת:

- מחלקה נקודה: מייצגת נקודה במישור, כולל מרחק מנקודה אחרת
- מחלקה פונקציה לינארית: מייצגת פונקציה מהצורה y=mx+k, בעלת בנאי משתי נקודות ושיטה לחישוב נקודת החיתוך בין שתי פוקנציות, וכן בדיקה האם קיימת נקודה כזו. המחלקה כוללת התייחסות לישרים אופקיים אנכיים ופונקצייה הופכית.
- מחלקה סגמנט: שמייצגת קטע במישור בין שתי נקודות, כוללת שיטה שמקבל נקודה ומחשבת האם הנקודה נמצאת על הסגמנט.
- מחלקה מלבן מקביל לראשית הצירים: עם שיטה שבודקת האם נקודה נמצאת בפנים, האם סגמנט חותך אותו – שיטה זאת ממומשת ע"י בדיקה של כל 4 הקטעים במלבן.
- מחלקה אוסף של מלבנים, שכוללת שיטה שבודקת האם קטע זר או לא לאוסף המלבנים ע"י בדיקה של כל אחד מהמלבנים.

- Class Point:
  - private double _x,_y;
  - public Point(double a,b);
  - public double distance(Point p);

- Class LinearFunction
  - private double _m,_k

- public LinearFunction(Segment t);
- private boolean _isVertical;
- public LinearFunction(Point p1, Point p2);
- public boolean isVerticel();
- public Boolean hasIntersection(LinearFunction l2);
- public Point intersection(LinearFunction l2);

- Class Segment
  - private Point _p1,_p2;
  - public Segmant(Point p1,Point p2);
  - public boolean isOn(Point q); // return true iff q is “close enough
    // to this Segment,uses EPS=0.01;
  - public boolean hasIntersection(LinearFunction l2);
  - public boolean hasIntersection(Segment s2);

- Class Rectangle
  - private Point _min, _max;
  - public Rectangle(Point p1, Point p2);
  - public boolean isInSide(Point q);
  - public boolean hasIntersection(Segment s2);

- Class Rectangles
  - private ArrayList<Rectangle>;
  - public boolean hasIntersection(Segment s2);

4.3 שיטות בדיקה:
ככלל נכתוב מחלקת בדיקה של Junit, של כל מחלקה בפרט, לפונקצייה לינארית, קטע, מלבן ואוסף מלבנים (לנקודה אין צרוך לכתוב בדיקות – יש לנו כבר רבות כאילו):

LinearFunctionTest
- שיטות בסיסיות לבדיקת הפונקציונליות של פונקציה לינארית לרבות חיתוך והתייחסות למקרים חריגים כמו פונקציות קבועות, אנכיות והפוכות.

SegmentTest
- שיטה לבדיקה של יצירה המרה לפונק׳ לינארית בדיקת חיתוך ובדיקה של ״האם נקודה נמצאת ע״ג סגמנט״, כולל חיתוך עם סגמנט אחר ופונ׳ לינארית.

RectangleTest
- שיטה לבדיקה בסיסית של היחס בין סגמנט פשוט למלבן פשוט כולל: בדיקת הכלה של אחד או שני הצדדים של הסגמנט, בדיקה של סגמנטים זרים מכל הצדדים וסגמנטים שחותך את המלבן בכל אחד מהצדדים.
- שיטה לבדיקה של סגמנטים מיוחדים לרבות: אנכי, אופקי, נקודתי למול מלבן יחיד.

RectanglesTest
- שיטה ליצירה של אוספים של מלבנים רלוונטי האוסף יכלול מלבנים חופפים ונפרדים. השיטה תופעל עם תג @Before, ככלל ניתן לממש זאת ע״י כתיבת מחלקה שעושה שימוש בתבנית עיצוב של Factory.
- שיטה לבדיקה של היחס בין סגמנט לאוסף גדול של מלבנים.
- שיטה שמייצרת מספר גדול של סגמנטים שכולים זרים לאוסף ובודקת זרות.
- שיטה שמייצרת מספר גדול של סגמנטים שכל אחד מהם אינו זר לאוסף ובודקת.

בהצלחה (גם למי שזקוק למועד ב׳) !!!