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

סמסטר א' 2019, מועד ב', 19.2.2019, מספר קורס: 7027910
אליזביט איצקוביץ, בעז בן משה.
מתרגלות: יעל לנדאו, אנה פרבר, רחל מוסטקיס

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 2-4 שאלות הבחינה

הקפידו על טוהר הבחינה!!

בהצלחה!

שאלה 1 (25 נקודות)

בשאלה זו נתייחס לממשק "פונקציה" שמייצג פונקציה רציפה בין הממשיים לממשיים

```
interface function{
        double f(double x); }
```

לדוגמא המחלקה Zero מייצגת את פונקציית האפס מחזירה לכל ערך x את הערך 0

```
class Zero implements function {
        public Zero() {;} // nothing to do.
        public double f(double x) {return 0;}
}
class Monom implements function{
    private double _a;
    private int _b;
    public Monom (double a, int b) {_a=a;_b=b;}
    public double f(double x) {
        double ans = _a * Math.pow(x,_b);
        return ans;
    }
}
```

1.1 (4 נקודות) כתבו את המחלקה RoundFunction שמייצגת את הפונקציה של "עיגול" משמע מקבלת מספר ממשי ומחזירה את המספר השלם הקרוב ביותר אליו.

```
class RoundFunction implements function{ //…}
```

1.2 (3 נקודות) השלימו את המחלקה Sum שמייצגת סכום שתי פונקציות מהצורה f1(x) + f2(x)

```
class Sum implements function{//…. }
```

1.3 (3 נקודות) השלימו את המחלקה Comp שמייצגת את הרכבה של שלי פונקציות מהצורה f2(f1(x))

```
class Comp implements function{//…. }
```

1.4 (15 נקודות) השלימו את המחלקה פולינום – שימו לב עליכם להשלים אך ורק את השיטות שהוגדרו – אין להוסיף שדות מידע.

```
class Polynom implements function {
        private function _ans;
        public Polynom() {_ans = new Zero();}
        public double f(double x) {return _ans.f(x);}
        public void add(Monom m) { // השלימו: השיטה שמוסיפה מונום לפולינום זה
        public void add(Polynom p) { // השלימו: שיטה שמוסיפה פולינום לפולינום זה
        public void sub(Polynom p) { // השלימו: שיטה שמפחיתה מפולינום זאת את p
        public void mul(Monom m) { // השלימו: שיטה שכופלת את הפולינום ב m
}
```

הצעת פתרון:

```
public class RoundFunction implements function{
        public double f(double x) {
                return (int) (x+0.5);}
        public String toString() {return "round(";} // לא נדרש
}
class Sum implements function{
        private function _f1, _f2;
        public Sum (function f1, function f2) {_f1=f1;_f2=f2;}
        public double f(double x) {return _f1.f(x) + _f2.f(x);}
        public String toString() {return ""+_f1+"+"+_f2;}// לא נדרש
}
class Comp implements function{
        private function _f1;
        private function _f2;
        public Comp (function f1, function f2) {_f1=f1;_f2=f2;}
        public double f(double x) {return = _f2.f(x) * _f1.f(x);}
```

```
        public String toString() {return "("+_f2+") ("+_f1+")";}  // לא נדרש
}

class Polynom implements function {
        public static final function _MINUS1 = new Monom(-1,0);
        private function _ans;
        public Polynom() {_ans = new Zero();}
        public String toString() {return _ans.toString();}  //לא נדרש
        public double f(double x) {return _ans.f(x);}
        public void add(Monom m) { _ans = new Sum(_ans,m);}
        public void add(Polynom p) { _ans = new Sum(_ans,p);}
        public void sub(Polynom p) {
                function mp = new Comp(_MINUS1, p);
                _ans = new Sum(_ans, mp); }
        public void mul(Monom m) { _ans = new Comp(m,_ans);}
}
```

דוגמא לmain - לא נדרש!

```
public class Main {
        public static void main(String[] args) {
                Monom m1 = new Monom(1.5,2);
                Monom m2 = new Monom(-3.5,4);
                Monom m3 = new Monom(3,0);
                Polynom p = new Polynom();
                p.add(m1);p.add(m2);p.mul(m3);
                System.out.println(p);
                double d = 2;
                System.out.println("p("+d+") = "+p.f(d));
        } // console output: (0+1.5x^2+-3.5x^4) (3.0x^0), p(2.0) = -150.0
}
```

<u>שאלה 2 (25 נקודות)</u>

בשאלה זו נבנה מערכת לפריצת קוד ע"י חיפוש כל האפשרויות.

הניחו שקיימת לכם מחלקה Password בעלת הפונקציה הסטטית

```
public static boolean testPass(String s) {\\..}\\returns true iff s is the password.
```

אתם רוצים למצוא את הסיסמא פשוט ע"י בדיקה של ססמאות שונות עד שתמצא הסיסמא.

הניחו שהסיסמא היא מחרוזת בת 8 ספרות (בין 0-9 – כולל חזרות)

2.1 (8 נקודות) השלימו את הפונקציה findPass שמחזירה את הסיסמא.

```
public static String findPass () {

        //…

} // findPass
```

2.2 (8 נקודות) הסבירו במילים כיצד ניתן לייעל את הקוד ע"י שימוש בשני תהליכים, פרטו כיצד תיעשה העבודה, ובאיזה אופן תוחזר התשובה מהפונקציה.

2.3 (9 נקודות) מומשו את הפונקציה כך שתעשה שימוש בשלושה תהליכים.

הצעת פתרון:

2.1

```
public class FindPass {
      private static int min=0, max=1000*1000*100, LENGTH=8;
      public static String findPass() {
            String ans = null;
            int ind = min;
            while(ans==null && ind < max) {
                  String test = getStringFromInt(ind,LENGTH);
                  if(Password.testPass(test)) { ans = test;} // done!
                  ind++;
            }
            return ans;
      } // findPass
      public static String getStringfromInt(int I, length l) {
            String s = “”+i;
            while(s.length()<l) {s=”0”+s;}
            return s;
      }
}
```

2.2

בעזרת שימוש בשני תהליכים ניתן לפרק את תחום החיפוש לשני תת תחומים שווים, נממש מחלקה שיורשת מ Thread שמקבלת תחום חיפוש, ובשיטה run היא תבצע בדיקה על כל אחד מספרים בתחום (ע״י הפיכה למחרוזת כמתואר ב 2.1). השיטה תסתיים כאשר נמצאה הסיסמא או שהסריקה הסתיימה. ברגע שאחד התהליכים מצא את הסיסמא נסיים גם את האחרים. בפתרון מלא ניתן לעשות שימוש ב wait, notify כדי להמתין למציאת הסיסמא מבלי לבדוק (פתרון זה לא נדרש כדי לקבל ציון מלא).

2.3

```
class FindPassBetween extends Thread{
      private int start, end;
      private boolean cont;
      private String pass = null;
      public FindPassBetween(int start, int end) {
            this.start = start;
            this.end = end;
            cont=true; pass=null;
      }
      public synchronized String getPass() {return pass;}
      private synchronized void setPass(String s) {pass=s;}

      public void run() {
            boolean flag = false;
            for (int i=start; cont && pass==null && i<end; i++) {
                  String test = FindPass.getStringFromInt(i,8);
                  flag = Password.testPass(test);
                  if(flag) {setPass(test);}
            }
      }
      public void terminate () {cont=false;}
```

```
public class FindPassThreeThreads {
      public static String findPadd3Th() {
            int n = 8, k = 10;
            int count = (int)Math.pow(k, n) - 1;
            int t1 = count/3, t2 = 2*t1;
            String pass=null;
            FindPassBetween process1 = new FindPassBetween(0, t1);
            FindPassBetween process2 = new FindPassBetween(t1+1, t2);
            FindPassBetween process3 = new FindPassBetween(t2+1,
count);
            process1.start(); process2.start(); process3.start();

      while(process1.getPass()==null&&process2.getPass()==null&&
process3.getPass()==null); // here some “sleep” or “wait” should be
done!
            process1.terminate(); process2.terminate();
process3.terminate();
            pass = process1.getPass();
            if(pass==null) {pass = process2.getPass();}
            if(pass==null) {pass = process3.getPass();}
            return pass;
      }
}
```

שאלה 3 (25 נקודות):

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

3.1 (8 נקודות) הסבירו מהו ממשק serializable של (java.io)

3.2 (8 נקודות) הסבירו בקצרה מהו ThreadPool, כיצד משתמשים בו ובאילו מקרים נכון לעשות זאת.

3.3 (9 נקודות) ציינו 4 תבניות עיצוב מרכזיות: עבר כל תבנית הסבירו בקצרה מה תפקידה ובאילו מקרים נכון להשתמש בה.

הצעת פתרון:

serializable הינו ממשק (ריק – ללא שיטות) בג'אווה אשר מממשים כאשר רוצים לסמן שאובייקטים ממחלקה נתונה יכולים לעבור תהליך סריאליזציה. סריאליזציה הוא תהליך של תרגום מבני נתונים או מצב של אובייקטים, לפורמט שניתן לאחסן אותו (לדוגמה בקובץ או להעביר אותו על גבי רשת מחשבים). משמע מיפוי של נתוני האובייקט לאוסף רציף של bytes.

3.2 ThreadPool זהו עיצוב תוכנתי בו יש ישות האחראית על ניהול ותפעול ה Threads הקיימים ופועלים במערכת. ThreadPool מאפשרת שימוש חוזר בתהליכונים קיימים לצורך ביצוע משימות עכשוויות ובכך פותרת את ה-overhead של המערכת הנגרם מיצירה והריסת תהליכונים. כמו כן, מגבילה את מספר התהליכונים הפועלים במערכת בכל רגע נתון - .

נשתמש ב ThreadPool כאשר נרצה להגביל את מספר התהליכונים הפועלים בהתאמה ליכולות תשתית המערכת שלנו. כמו כן, נשתמש כאשר יש שימוש חוזר ומרובה במשימות מקביליות לביצוע בכדי למטב ביצועים ולעשות שימוש חוזר בתהליכונים קיימים.

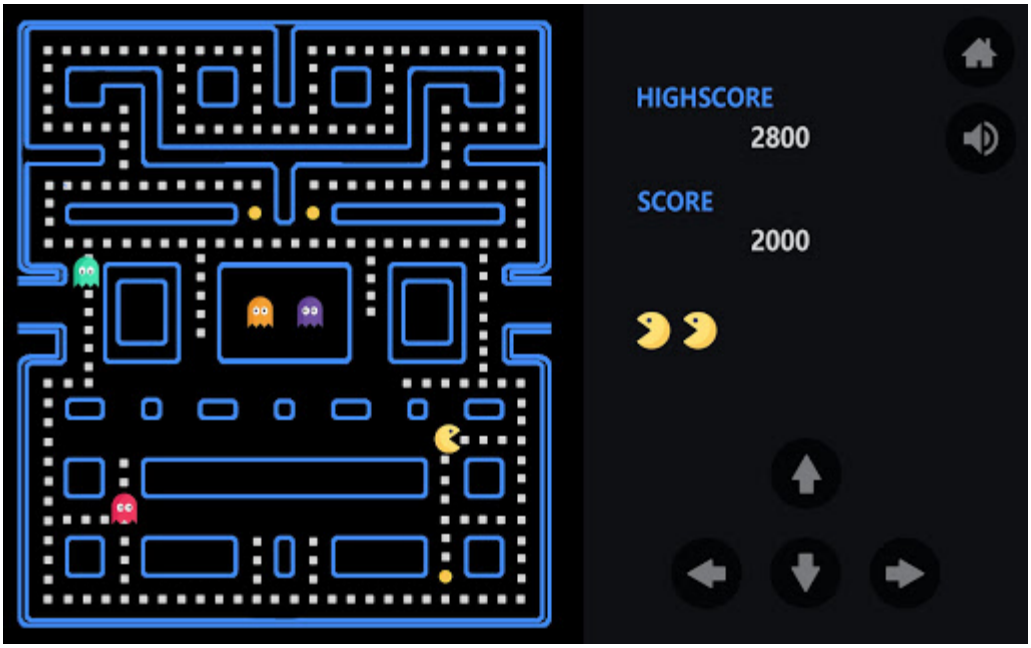
3.3

- **Singleton** כאשר רוצים לשלוט על מספר המופעים אותם ניתן ליצור ממחלקה מסוימת. בפרט (אבל לא חובה) כאשר נרצה לחייב שלמחלקה מסוימת יהיה אובייקט יחיד. הגדרת נקודת כניסה יחידה לאובייקט לשם הקצאת מופע יחיד של האובייקט בזיכרון.נשתמש כאשר: חייב להיות מופע אחד בדיוק של מחלקה, וחייבת להיות נקודת ממשק ידועה למחלקה זו.

- **Iterator** מאפשרת מעבר על איברי אובייקט נתון באופן שאינו תלוי במימוש הפרטני של מבנה הנתונים בו משתמש האובייקט. הממשק מעבר סידרתי (כולל בדיקת סיום ומחיקה) על מבנה נתונים ללא תלות במימוש הקונקרטי שלו. ובכך יוצרת אי תלות בין המימוש של מבנה הנתונים והאלגוריתמים שמשתמשים בו.
- **Facade**: מאפשר ממשק אחיד לאוסף של ממשקים בתת-מערכת. Facade מגדיר ממשק ברמה גבוהה יותר המאפשר שימוש קל בתת-מערכת. נשתמש כאשר: נרצה לתת ממשק פשוט לתת מערכת מורכבת. קיימים יחסי תלות רבים בין הלקוחות והמחלקות המממשות. Facadeמסייע בהחלשת הצימוד בין הלקוחות ותתי המערכות. כאשר רוצים לעבוד בשכבות בתתי המערכת, כאשר ה Facade-הוא נקודת הכניסה לרמת התת-מערכת. אם תתי המערכות הן בלתי תלויות, ניתן לפשט את התלות בינן על ידי הקפדה על כך שכל התקשורת בינן תעשה דרך ה Facade-בלבד.
- **Decorator** כאשר רוצים להוסיף פונקציונליות מסוימת לאובייקטים מבלי לשנות את המבנה הקיים שלהם. הוספת אחריויות נוספות לאובייקט באופן דינמי תוך שמירה על אותו הממשק Decorator. משמש כחלופה גמישה לתת מחלקה כדי להרחיב פונקציונליות. נשתמש כאשר: נרצה להוסיף אחריות באופן דינמי ושקוף לאובייקטים מסוימים (בלא להשפיע על אובייקטים אחרים).

כמובן יש תבניות עיצוב רבות ומגוונות נוספות שמחולקות לשלוש קבוצות עקריות:
יצירה, מבנה, התנהגות.

<u>שאלה 4 (25 נקודות):</u>
בשאלה זו נתייחס לתכנון של "אסטרטגיית רדיפה" של "מפלצות" אחרי packman.
ברוח מטלה 4: עליכם לתכנן מחלקה שתזיז את ה"מפלצות" ע"ג זירת משחק כדי לאפשר לכם לתפוס את ה packman כמה שיותר מהר.
הדרכה: בשאלה זו אין צורך לכתוב קוד ואפשר להניח שהמחלקות שעוסקות בניהוג השחקן (ה packman) קיימות.
הניחו שאתם מודעים למיקום השחקן (ה packman – היחיד) ולמהירות שלו (אשר גבוהה מהמהירות של כל אחת מה"מפלצות" פי שתיים), וכן שאתם מודעים לגבולות הזירה ממנה לא ניתן לצאת, ושמטרת הpackman היא לאסוף את כל הפירות בזירה. הניחו שבתחילת המשחק הpackman ממוקם בגבולות הזירה, וכל המפלצות ממוקמות במרכז הזירה.
4.1 (8 נקודות) הסבירו באופן כללי כיצד תממשו אסטרטגיה עבור מפלצת יחידה שרודפת אחרי שחקן יחיד – בזירה ללא מכשולים כלל. הסבירו כיצד ניתן לייצר "רמות קושי" שונות.
4.2 (8 הנקודות) הרחיבו את הפתרון עבור שתי מפלצות ושלוש מפלצות בזירה עם מכשולים.
4.3 (9 נקודות) רשמו את המחלקות העיקריות אותן יש לממש כדי לאפשר פיתוח של מערכת כנל, לכל מחלקה ציינו את השיטות העיקריות שלה ואילו שדות מידע משמעותיים יש לה.



הצעת פתרון

4.1 אסטרטגיית הרדיפה של המפלצת יחידה אחרי ה packman תמומש כך:

- גרסה פשטנית: המפלצת תחשב כל פעם את הכיוון לpackman ותלך לשם.
- בגרסה משופרת המפלצת "תחשב" (תשערך) לאן ה packman מתכנן ללכת (לאיזה פרי, באיזה סדר) ואז תנסה להגיע למסלול המשוער של ה packman.
- גרסה קצת יותר מתקדמת: המפלצת תעבור על כל הפירות שמשחק ותמצא את קבוצת הפירות שהמפלצת יכולה להגיע אליהם לפני ה packman, ותבחר מתוכם את הפרי הכי "קרוב" למפלצת ותלך "לשמור עליו".

4.2 אם יש שתיים, שלוש מפלצות ומכשולים נוכל לממש אסטרטגיה דומה לזו שהוגדרה בסעיף הקודם, אבל הפעם מדידת המרחקים תהיה דרך גרף של מרחקים מעל המכשולים.
בגרסה מתקדמת יותר ישתפו המפלצות פעולה בכך שיבחרו את האזור שבו ניתן לבצע "חסימה" של נתיב התנועה של packman בעזרת 2,3 מפלצות.

4.3 המחלקות הרלוונטיות למימוש הן:

- נקודה דו מימדית, כוללת שיטה לחישוב מרחק מנקודה אחרת.
- מסלול (אוסף סדור של נקודות).
- מפלצת: כוללת מידע לגבי מיקום, מהירות, ושיטה לשערוך הזמן שייקח למפלצת להגיע לנקודה נתונה.
- אוסף של מפלצות
- Packman מייצג "שחקן" בעל מיקום, מהירות ולוגיקת תנועה.
- מכשול – מייצג מלבן מקביל לראשית הצירים – שלא ניתן לנוע דרכו
- אוסף מכשולים.
- זירה: כוללת אוסף של מפלצות, שחקן (packman), ואוסף מכשולים
- אלגוריתם לתנועה של מפלצת יחידה – לפי רמות.
- אלגוריתם לתנועה של אוסף מפלצות – לפי רמות.

בהצלחה!!!