# **מבחן ב-** "תכנות מונחה עצמים"

סמסטר א' 2018 – מועד א' 4.2.2018
אראל סגל-הלוי, אודי לביא, בעז בן משה
תשעח_א_7027910

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

במבחן זה 4 שאלות, יש לענות על כולן.

אנא רשמו את תשובותיכם במחברת התשובות בלבד.
הקפידו לרשום בדף התשובות גם את מספר הנבחן ותעודת הזהות שלכם.

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)**

נתונות המחלקות והממשקים הבאים:

1. ממשק פונקציה (function) – שמייצג פונקציה רציפה מהממשיים לממשיים
2. המחלקה מונום שמייצגת פונקציה מהצורה f(x) = a*x^b כאשר a הוא ממשי ו b הוא מספר שלם אי שלילי.
3. מחלקה שמממשת את ממשק ה Comparator<Monom> אשר מגדירה יחס סדר קווי על מונומים לפי החזקה שלהם.

```
/**This interface represents a function y=f(x), where f(x) and x are reals. **/
public interface Function{
        public double f(double x);}

public class Monom implements Function{
        public Monom(double a, int b){...}
        public Monom(Monom ot) {...}
        public int get_power() {...}            // מחזיר את החזקה של המונום
        public double get_coefficient () {...}          // מחזיר את המקדם של המונום
        public double f(double x) {...}
        public void add(Monom m)  {...}             // הוספת מונום למונום זה
        public void multiply(Monom m)  {...}        // כפל מונום במונום זה
        public boolean equals(Monom m){...}
}
public class Monom_Comperator implements Comparator<Monom>{
        public int compare(Monom m0, Monom m1){
                int t = m1.get_power() - m0.get_power();
                return t;
        }
}
```

1.1 (6 נקודות) כתבו מחלקה שמממשת הממשק Comparator<Function> ומקבלת ערך ממשי בבנאי (נניח x0) ומשווה בין שתי פונקציות לפי ערכן בנקודה x=x0.
הדרכה: בעזרת המחלקה הזאת נוכל למיין אוסף פונקציות לפי ערכן בנקודה מסויימת (מגדול לקטן).

1.2 (6 נקודות) ממשו את השיטה **מונום הנגזרת**: public Monom derivative()
(ניתן להניח שכל שאר השיטות שרשומות מעלה קיימות עבורכם).

1.3 (6 נקודות) כתבו מחלקה שמייצגת אוסף פונקציות (<Function>): המחלקה תאפשר להוסיף פונקציה לאוסף, להחזיר את גודלו (כמות הפונקציות), ולקבל את הפונקציה במקום מסויים (לפי אינדקס)

public class ListOfFunctions…

1.4 (7 נקודות) כתבו את מחלקה שמייצגת את הממוצע של אוסף פונקציות

public class AveOfFunctions implements Function {

למחלקה יש בנאי שמקבל אובייקט מסוג ListOfFunctions

**תשובה:**

1.1

```
public class Function_Comperator implements Comparator<Function>{
	private double _x0;
	public Function_Comperator(double x0) {_x0=x0;}
	public int compare(Function f0, Function f1){
		int ans = 0;
		double df = f1.f(_x0) - f0.f(_x0);
		if(df>0) {ans = 1;}
		if(df<0) {ans=-1;}
		return ans;
	}
}
```

1.2

```
public Monom derivative() {
	double p = this.get_power();
	double c = this.get_coefficient ();
	return new Monom(p*c,p-1);
}
```

1.3

```
public class ListOfFunctions{
	private ArrayList<Function> _functions;
	public ListOfFunctions() {_faunctions = new ArrayList<Function>();}
	public int size(){return _functions.size();}
	public void add(Function f) {_functions.add(f);}
	public Function get(int i) {return _functions.get(i);
}
```

1.4

```
public class AveOfFunctions implements Function{
	private ListOfFunctions _fs;
	public AveOfFunctions(ListOfFunctions fs) {_fs = fs;}
	public double f(double x) {
		double ans = 0, s = _fs.size();
		if(s==0) {throw new RuntimeException("Error: empty list!!");}
		for(int i=0;i<s;i++) {ans+=_fs.get(i).f(x);}
		ans = ans / s;
		return ans;
	}
}
```

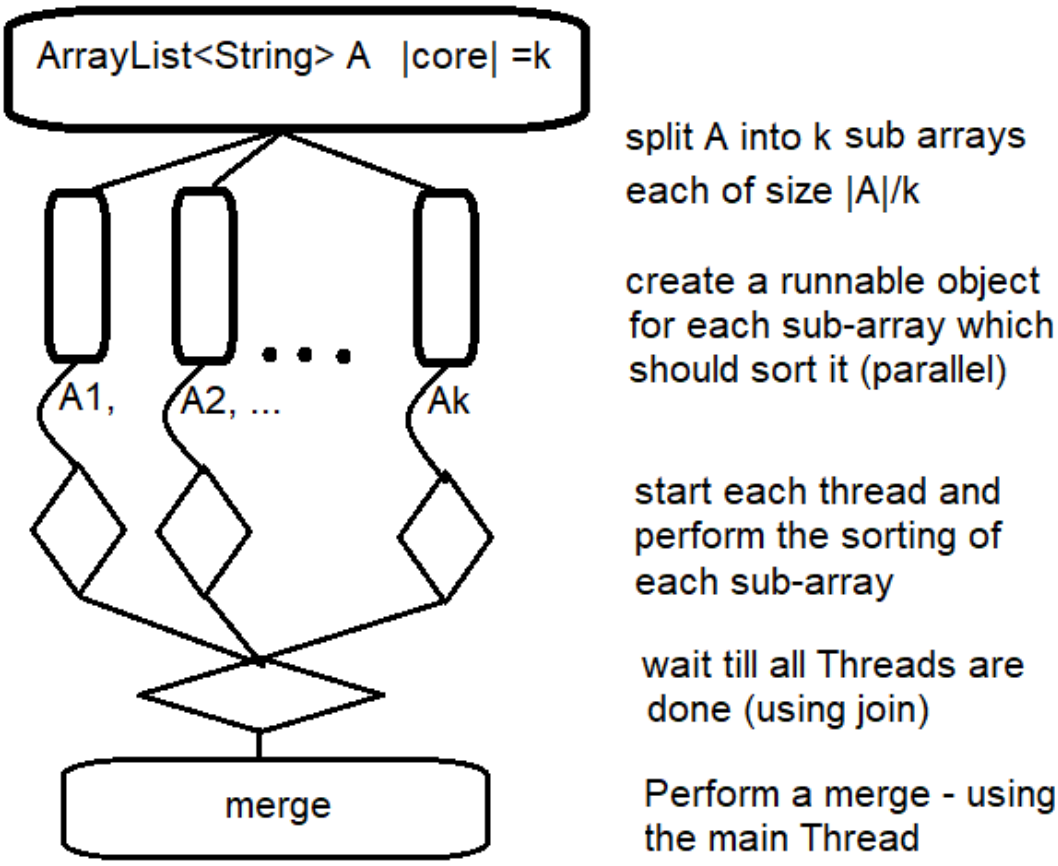
**שאלה 2 (25 נקודות)**

בשאלה זו נתייחס למיון אוסף מחרוזות שנמצאות ב ArrayList <String>.
הסבירו באופן כללי (בטקסט ואו בתרשים) כיצד ניתן בעזרת ריבוי תהליכונים (Threads) לייעל את פעולת מיון המחרוזות (נניח שיש לנו Comparator<String>). תארו את מבנה המערכת ואופן הפעולה שלה.

**הדרכה**: בשאלה זאת אין צורך לכתוב קוד אלא הסברים + דיאגרמות UML בלבד.

**תשובה:** (מבוססת על הפתרון שסביר שזכה במלא הנקודות).

- נחלק את מערך הקלט לחלקים בהתאם למספר הליבות במערכת. למשל אם יש 4 ליבות ואורך המערך הוא 1000, אז החלקים יהיו באינדקסים: 0 עד 249, 250 עד 499, 500 עד 749, 750 עד 999.
- עבור כל חלק, ניצור חוט (Thread) אחד שאחראי למיין אותו. החוט ישתמש באלגוריתם מיון כלשהו או במיון של ג'אבה: Collections.sort.
- נחכה שכל החוטים יסיימו את עבודתם (ע"י join). לאחר מכן, נאחד את המערכים הממויינים לתוך מערך הפלט, תוך שימוש באלגוריתם "מיזוג" (כמו במיון-מיזוג).




תרשים UML מומלץ (לא חובה).

פתרון נוסף שזכה במלוא הנקודות היה שימוש בזרמים - Streams: הופכים את המערך לזרם (stream), הופכים אותו למקבילי (parallel), ומסדרים אותו (sort). פתרון זה אף עדיף על הפתרון הראשון - הוא יותר קצר, יותר יעיל וחוסך טעויות.

**שאלה 3** (25 נקודות):

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

3.1 (6 נקודות) הסבירו בקצרה מהו Threadpool, ציינו מקרה שבו כדאי להשתמש ב ThreadPool.

תשובה: ThreadPool היא שיטה לבצע הפשטה בין כמות התהליכים הנדרשים למימוש האלגוריתם לבין כמות התהליכים שרצים בזמן אמת על המחשב בגלל אילוצי משאבים (לרוב זיכרון, משאבי חישוב ליבות פיסיות וכו'). וכך המתכנת יכול להגדיר כמה תהליכים שצריך בעוד שהמערכת (המחלקה Threadpool) דואגת שכמות התהליכים שרצים באמת בצורה מקבילית תואמת את הגדרות "משאבי המערכת". דוגמא פשוטה לכך ניתן לראות בשאלה 2 – כאשר ניתן להגדיר בצורה פשטנית את כמות התהליכים להיות ככמות הליבות הפיסיות שזמינות לביצוע המיון.

3.2 (6 נקודות) מהו ה Design Pattern – Singleton, הסבירו בקצרה מתי נכון להשתמש בו וכיצד ניתן לממש אותו.

תשובה:Singleton היא שיטה להגדרה של מחלקה שממנה ניתן "לייצר" אך ורק אובייקט יחיד – שימוש בשיטה זאת נעשה כאשר רוצים להבטיח אובייקט יחיד – כגון אובייקט פרמטרים נוכחי של משתני מערכת. או כל משאב שיש שמייצג אובייקט שממנו יש עותק יחיד ולא נכון לאפשר לייצר ממנו יותר מאובייקט אחד. מימוש שיטה זאת נעשה בד"כ ע"י כך שהבנאי של המחלקה הוא פרטי, ויש פונקציה סטטית שמחזירה עותק יחיד מהאובייקט – אם קיים מחזירה פשוט את המצביע אליו ואם לא מייצרת עותק סטטי יחיד ומחזירה אותו.

3.3 (6 נקודות) סטודנט מימש מחסנית ע"י כך שהוא ירש מהמחלקה של ArrayList, הסבירו איזו בעיה יש בפתרון כזה מבחינת עקרונות תכנות "מונחה עצמים", הסבירו כיצד ניתן לפתור בעיה זאת.

תשובה: הבעיה היא שהמחסנית שהסטודנט מימש כוללת כעת גם שיטות שאינן רלוונטיות למחסנית כגון – גישה למקום מסויים, מחיקה\הוספה של איברים בכל אינדקס. ניתן לפתור זאת ע"י כך שלא נירש מ ArrayList אלא פשוט נשתמש בו כ data member – ואז נממש אך ורק את השיטות שרלוונטיות למחסנית עליו.

3.4 (7 נקודות) כתבו בקצרה מהן הפעולות המרכזיות שעושים ב git, ציינו מספר יכולות שקיימות ב github ואינן חלק מ git.

תשובה: הפעולות המרכזיות ב git הן הפעולות של כל מערכת בקרת תצורה לרבות:

1. יצירת repository
2. הוספה של קבצים \ תיקיות ל repository (וכן מחיקה של קבצים)
3. ביצוע עדכון – commit
4. יצירת "ענף" branch וכן איחוד בעזרת merge
5. הגדרה של גרסה (Head) לפי version / commit

ב github יש יכולות נוספות מעבר ל git בפרט נושאים שקשורים לניהול פרויקט: חלוקת משימות, ניהול issues

**שאלה 4 (25 נקודות):**

בשאלה זו נניח שקיימת מחלקה מתמטית בשם MyMath ובה יש פונקציה מתמטית שמחשבת לכל מחרוזת מספר שלם (long) – הניחו שעבור קלט נתון הפונקציה f תמיד מחזירה אותו פלט.

```
public class MyMath() {
        public static long f(String s) {…} //  פונקציה קיימת

}
```

4.1 (10 נקודות) הסבירו במילים כיצד ניתן לגרום לכך שהפונקציה f לא תרוץ כל פעם מחדש כאשר היא נקראת עם אותו קלט.

4.2 (15 נקודות) הוסיפו למחלקה את הקוד הנדרש בהתאם (כך שהפונקציה f לא תרוץ כל פעם מחדש כאשר היא נקראת עם קלט עליו היא רצה כבר).

**תשובה:**

4.1

נעשה שימוש בזיכרון (מפה) נכתוב פונקציה ש"עוטפת" את הפונקציה המקורית כאשר בכל פעם שהפונקציה החיצונית נקראת תתבצע בדיקה האם כבר חושבה הפונקציה על הקלט הנ"ל אם לא תקרא הפונקציה המקורית והערך שלה ישמר (במיפוי של קלט,פלט), אם כן פשוט נחזיר את הערך של הממופה של הקלט.

מבנה הנתונים שמתאים לכך הוא HashMap<String,Long>

4.2

```
public class MyMath {
      private static long f1(String s) {…} // פונקציה קיימת
      public static long f(String s) {
            Long ans =  _memory.get(s);
            if(ans == null) {ans = f1(s); _memory.put(s, ans);}
            return ans;
      }
      private static Map<String, Long> _memory =
                                          new HashMap<String,Long>();
}
```

בהצלחה!!!