המסלול האקדמי
המכללה למינהל

להצלחה יש דרך

מבחן לדוגמא

ד"ר אליהו חלסצ'י
eliahukh@colman.ac.il

הקדמה

במבחן זה עליכם לענות על 4 \ 4 שאלות תכנותיות ב JAVA. משך המבחן 3 שעות. חומר פתוח. עליכם להקפיד היטב על ההוראות, ובפרט על הוראות ההגשה, שכן הבדיקה הינה אוטומטית.

אתם מקבלים:

- את קובצי המקור אותם עליכם להשלים
- Main לבדיקה לוקאלית

עליכם להגיש

- את קובצי המקור מושלמים. לא ב zip או דומיו, אלא את קובצי המקור עצמם.

קוד שלא מתקמפל או שיש לו שגיאות בזמן ריצה יקבל הפחתה אוטומטית של 20 נקודות.

עליכם להקפיד על ה API הנדרש, הבדיקות של ה Main הלוקאלי, וכמובן על הוראות התרגיל, שכן ה Main הלוקאלי בפירוש לא בודק את כל המקרים שכן נבדקים במערכת האוטומטית של המבחן.

מבנה המבחן:

- שאלת מימוש קוד מתוך תרשים UML
- שאלת Design Patterns והרכבות שונות מוכוונות עצמים
- שאלת Concurrency Design Patterns והרכבות שונות
- שאלת Architectural Patterns ו\או שאלות כלליות על החומר

עליכם להגיע לפני תחילת המבחן ולהתיישב בעמדות שבמעבדה. במידה והנכם רשומים למעבדה 36/7 אנא שבו בצד של המעבדה אליה אתם רשומים. עם תחילת המבחן יתפרסם טופס המבחן במודול של הקורס. האינטרנט ינותק והמבחן יתחיל. מי שמסיים לפני הזמן יכול להגיש את המבחן למערכת הבדיקות, שימו לב שניתן להגיש רק פעם אחת, ולא יינתן פידבק מידי.

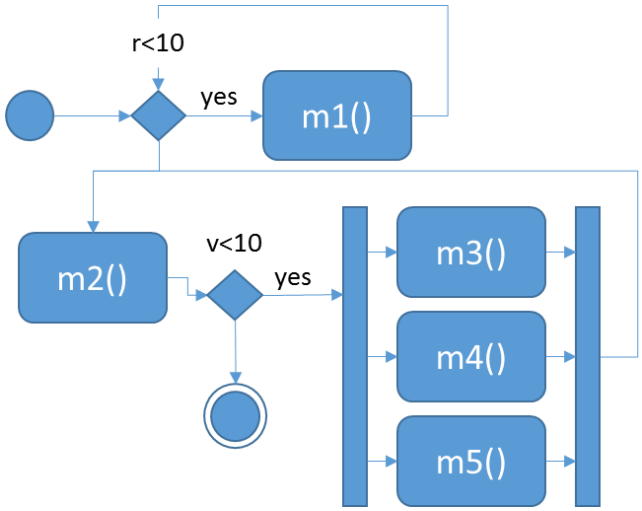
בהצלחה!

אלי

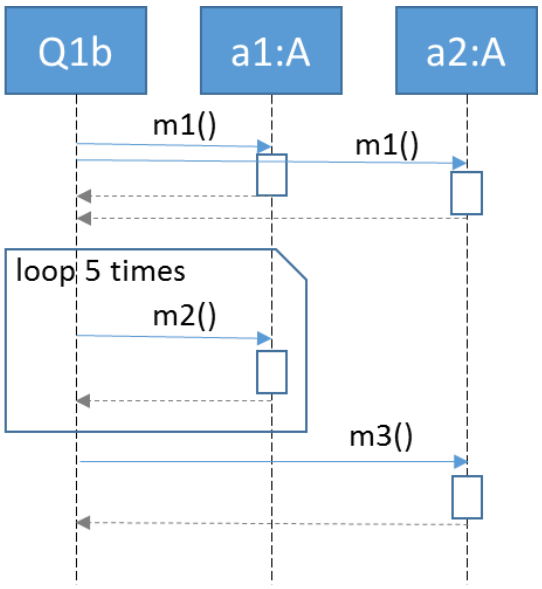
שאלה 1: (25 נק')

נתונה לכם המחלקה A עם המתודות m1() עד m5(). במחלקה Q1 השלימו את המתודות a() , b() ו c() ע"פ סעיפי שאלה 1. עליכם להפעיל את המתודות של A ע"פ תרשימי ה UML הבאים:

סעיף א' (10 נק'):



סעיף ב (10 נק'):



סעיף ג' (5 נק'):

האפשרות ליצור איזה סדר קריאה שנרצה לאלו מתודות של A שרק נרצה, מתאר היטב את עקרון ה:

- IOC
- SOC
- DIP
- DI

החזירו את המחרוזת המתאימה במתודה c() של המחלקה Q1.

שאלה 2: Design Patterns (20 נק')

השלימו את הקוד של המחלקה Task, כך שיהיה ניתן להפעיל משימה כלשהי, וכן להגדיר אלו משימות ירוצו כשהמשימה הזו תסתיים.

```
public abstract class Task ____________________________________{

	abstract void theTask();

	// triggers another task on completing the task
	public void triggers(Task t){
		_______________________
	}

	public void run(){
		theTask();
		_______________________
	}

	_____________________________________
	_____________________________________
	_____________________________________

}
```

לדוגמא:

```
		Task t1=new Task() {
			@Override
			void theTask() {
				System.out.println("taks1...");
			}
		};
		Task t2=new Task() {
			@Override
			void theTask() {
				System.out.println("taks2...");
			}
		};
		Task t3=new Task() {
			@Override
			void theTask() {
				System.out.println("taks3...");
			}
		};

		t1.triggers(t2);
		t2.triggers(t3);

		t1.run();
```

הפלט יהיה:

```
taks1...
taks2...
taks3...
```

שאלה 3: Concurrency Patterns (40 נק')

ברצוננו ליצור מנגנון מסוג Active Object שאליו יהיה ניתן להגדיר תחת מפתחות שונים פונקציות שונות (פונקציה - בהינתן ערך כלשהו מסוג P תחזיר ערך שיכול להיות מטיפוס כלשהו אחר R).

לדוגמא:

```
GenericFunctionActiveCaller gfc=new GenericFunctionActiveCaller();

gfc.addFunction("mul2", (Double x)->x*2);

gfc.addFunction("print", (Object x)->{
       System.out.println(x.toString());
       return x.toString();});

gfc.addFunction("sqr", (Double x)->x*x);

gfc.addFunction("len", (String x)->x.length());
```

- הקריאה לפונקציה לא תהיה סינכרונית, אלא תתבצע בת'רד של ה Active Object שלנו.
- מכיוון שזה יכול לקרות בעוד זמן רב, הקריאה תחזיר מיד Future שיכיל את ערך החזרה,
  - מתודת ה ()get של ה Future תממש guarded suspension.
- קריאה למתודה stop תעצור את ה Active Object ואת כל משימותיו.

דוגמא לשימוש:

```
gfc.exec("print",gfc.exec("mul2", gfc.exec("sqr",2.0).get()).get());
```

התוצאה תהיה הדפסה של 8.0, שימו לב שבדוגמא לעיל ביצענו המתנה לתוצאה.

השלימו את הקוד הבא כדי לעמוד בדרישות לעיל. הערה: אין להשתמש בקוד קיים של java.util.concurrent אלא רק בקוד שתממשו בתוך המחלקה הבאה.

```
public class GenericFunctionActiveCaller {

       public interface F<____>{ ___ apply(____);}

       public class Future<V>{
              V v;
              public _______ void set(V v){ השלימו את הקוד }
              public V get(){   השלימו את הקוד }
       }


       HashMap<String,_____> commands;
       BlockingQueue<Runnable> dispatchQueue;
       _________ boolean stop;
       Thread t;

       public GenericFunctionActiveCaller() { השלימו את הקוד }

       public ____ void addFunction(String key, _____){השלימו את הקוד  }

       public _____ Future<__> exec(String key,___ params){השלימו את הקוד}

       public void stop(){השלימו את הקוד}
}
```

שאלה 4 (15 נק'):

ענה על הטענות הבאות נכון \ לא נכון ע"י החזרה של true או false (בהתאמה) במתודות q1 עד q3 של המחלקה Q4 בהתאמה לסעיפים א-ג. כל סעיף 5 נק'.

א. Proxy Pattern שונה מ Object Adapter בכך שהוא מקבל אובייקט מסוג קונקרטי, ולא מסוג אבסטרקטי.

ב. ניתן להשתמש ב Bridge כדי להפריד בין data לבין functionality כך שכל מימוש של ה functionality יכול לפעול על כל אובייקט מהסוג של ה data.

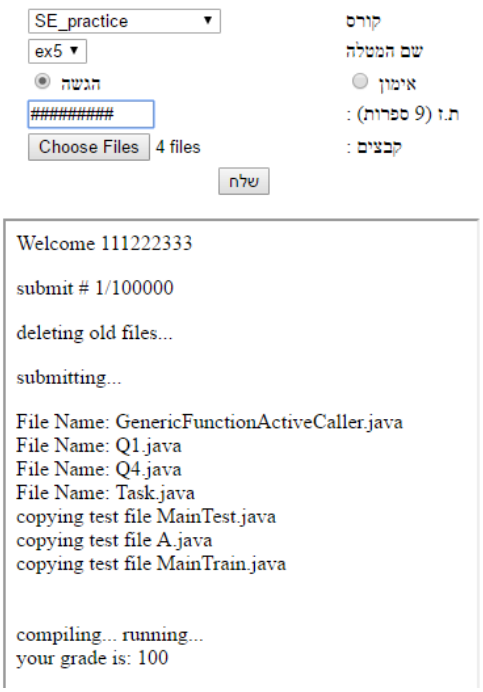
ג. אחד היתרונות של Pipeline & filters הוא שניתן לראות את סדר העיבוד בצורתו הלינארית שלו לעומת לולאות מקוננות וקוד מסובך.

הגשה

פרט לטופס מבחן זה קבלתם גם קובצי קוד מקור נוספים:

- GenericFunctionActiveCaller.java, Q1.java, Q4.java, Task.java
- A.java, MainAPItest.java

את הקבצים שבשורה הראשונה עליכם לערוך ע"פ הגדרות השאלות שבמבחן. אין להוסיף מחלקות או לשנות דבר כלשהו ב API. על כל המחלקות להיות ב package בשם test.



הקובץ MainAPItest.java נועד לשימוש לוקאלי שלכם בעת המבחן, והוא רק בודק API. מטרתו לעזור לכם להימנע משגיאות קומפילציה וריצה. מותר ואף רצוי במהלך המבחן לשנות אותו ולהוסיף לו בדיקות. מותר לכם גם לשנות את מחלקה A (אך לא את ה API שלה).

לכשתסיימו עליכם להגיש את קובצי המקור שבשורה הראשונה ואותם בלבד, למערכת הבדיקה האוטומטית. מכיוון שזה מבחן לדוגמא, אין הבדל בין מוד "אימון" ל"הגשה", אך ברור שיש שם בדיקות אמתיות ולא רק בדיקות API כמו שב MainAPItest. כדי לצלוח אותן יש להקפיד על הוראות המבחן.

כמו כן, מכיוון שמדובר במבחן לדוגמא, תוכלו להגיש כמה פעמים שתרצו ולקבל משוב מידי. במבחן האמתי המשוב היחיד שתקבלו הוא האם היו שגיאות ריצה או קומפילציה, ובמידה והיו תוכלו להגיש שוב. מעבר לכך ישנה אפשרות להגיש רק פעם אחת עד תום המבחן.

בנוסף, תתבקשו להגיש את התשובות שלכם למבחן לשרת גיבוי. ובכלל, כדאי לשמור אצלכם את התשובות שלכם למבחן לכל מקרה.

את המבחן לדוגמא תוכלו להגיש ל SE_practice תחת ex5

בהצלחה!