תכנות מתקדם 2: 89-211 – מבחן לדוגמא

זמן המבחן: **שעתיים וחצי**, יש לענות על 4 מתוך 4 שאלות, **בגוף השאלון בלבד**. חומר **סגור**.

שתי שאלות בקיאות (מצריכות מכם להקיף את החומר) + שתי שאלות עיצוב קוד ותכנות (#c \ java).

בקיאות

שאלה1: (20 נק') לאפליקציית ה web שלנו יש בעיית סקלביליות.

- אהרון טוען שיש לבצע scale out, ואז load balancing באמצעות אלג' round robin
  - כלומר הטלת המשימות (טיפול בבקשות הלקוחות) על השרתים תתבצע ע"פ סבב קבוע.
- ברכה טוענת שיש לבצע scale up, ומכיוון שמדובר ב sticky session עם הלקוח אז ממילא חלוקת המשימות צריכה להתבצע באמצעות session affinity.
  - כלומר אותו השרת שטיפל בלקוח מסוים ימשיך לטפל גם בבקשות הבאות שלו, ולכן אין צורך ב scale out.
- גדעון לא מסכים עם ברכה. הוא לא מאמין ב session affinity והוא טוען שיש צורך ב scale out.
  - כל שרת יוכל לטפל בכל משימה שתהיה, כי את ה data של ה session נשמור בשרת אחר המהווה central session store, וכך נוכל לבצע load balancing טוב יותר!
- דקלה מציעה גם scale out וגם session affinity באופן הבא:
  - כבר בבקשה הראשונה נשמור ב cookie אצל הלקוח את ה ID של ה session. בכל בקשה נוספת מהלקוח הוא יעביר את המידע שב cookie וכך נוכל להקצות את המשימה אל השרת שטיפל בו בעבר.
- הרשלה טוען שיש לבצע scale out, ו IP Address Affinity.
  - בהינתן בקשת לקוח, נריץ hash על ה IP שלו, והתוצאה מודולו N תקבע מי מתוך N השרתים צריך לטפל בבקשה שלו.

עבור על אחד תארו בקצרה מהם היתרונות והחסרונות בשיטה שהציע (4 נק' לכל שיטה).

אהרון יתרונות:_____

_____

חסרונות: _____

_____

ברכה יתרונות:_____

_____

חסרונות: _____

_____

גדעון יתרונות:_____

_____

חסרונות: _____

דקלה יתרונות:

חסרונות:

הרשלה יתרונות:

חסרונות:

שאלה 2: (12 נק')

הקיפו בעיגול את התשובות הנכונות:

א. ניתן באמצעות double check locking להתגבר על בעיית ה singleton בסביבה שהיא multithreaded.
ב. MVVM ניתן ממש רק ב NET. בטכנולוגיית WPF.
ג. Service Locator מהווה Runtime linker
ד. קיים מימוש ל MVC בו שכבות המודל וה View צריכות להכיר רק את עצמן.

שאלה 3 (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){

		_____________________________________________________________

		_____________________________________________________________
		}
		public V get(){

		_____________________________________________________________

		_____________________________________________________________

		_____________________________________________________________

		_____________________________________________________________

		_____________________________________________________________

		_____________________________________________________________

		_____________________________________________________________

		_____________________________________________________________
		}
	}

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

	public GenericFunctionActiveCaller() {

		_____________________________________________________________

		_____________________________________________________________

		_____________________________________________________________
	}
```

```
        public ________ void addFunction(String key, _____){

                ____________________________________________________________

                ____________________________________________________________

                ____________________________________________________________
        }

        public _____ Future<____> exec(String key,_____ params){

                ____________________________________________________________

                ____________________________________________________________

                ____________________________________________________________
        }

        public void stop(){

                ____________________________________________________________

                ____________________________________________________________
        }
}
```

שאלה 4: (28 נק')

סעיף א' (20 נק')

נתון לכם הפסאודו-קוד הבא, השלימו את הקוד הבא כדי לממש את האלגוריתם בצורה מכוונת עצמים. על המימוש להפריד בין האלגוריתם לבין הבעיה שהוא פותר.

Selection_sort(list of n elements)

1. For i=0..(n-1)
2. currentMin = list[i]
3. newMin = minimum value in the list from index i+1 to n
4. If currentMin != newMin then swap them

השלימו:

```
public interface Sorter { // 2 points

        <E> void sort(_____________ sortable);
}
public interface Sortable<E> { // 3 points

        ___________________________

        ___________________________
        void swap(int i,int j);
        int size();
}
```

```
public class SelectionSorter implements _______________{

        @Override
        public <E> void sort(_____________ sortable) { // 15 points


        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        ____________________________________________________________

        }
}
```

סעיף ב' (8נק') – בהינתן LinkedList של אלמנטים כלשהם, ממשו בקוד עיצוב המאפשר לכם למיין את הרשימה ע"י הקוד שיצרתם בסעיף א'. הקפידו על חשיבה קדימה, חסכון בקוד ותכנות גנרי.

```
public class ListSortable<E> implements ___________________{

_____________________________________________________

_____________________________________________________

_____________________________________________________

_____________________________________________________
```

```
    public ListSortable(_______________________________) {

        ____________________________________________________

        ____________________________________________________
    }

    @Override
    public void swap(int i, int j) {

        ____________________________________________________

        ____________________________________________________

        ____________________________________________________

        ____________________________________________________
    }

    @Override
    public int size() {

        ____________________________________________________

    }

    ________________________________________________

    ________________________________________________

    ________________________________________________

    ________________________________________________

    ________________________________________________

    ________________________________________________

    ________________________________________________

    ________________________________________________

    ________________________________________________

    ________________________________________________

}
```