תכנות מתקדם 2: 89-211 – מועד א' תשע"ו

זמן המבחן: **שעתיים**, יש לענות על 6 מתוך 6 שאלות, **בגוף השאלון בלבד**. חומר **סגור**.

בקיאות

שאלה1: (16 נק')

נתונה לנו מערכת לניהול אותות מצוקה (signals) עבור קבוצת רובוטים. במחלקה SignalManager קיימת המתודה ()sendAck שפועלת באופן הבא: כל עוד אות המצוקה נקלט - המשך לשלוח Ack לרובוט שבמצוקה.

אליס העלתה שקיים חשש ש ()sendAck תהיה מתודה חוסמת במקרה וה Ack לא נקלט אצל הרובוט שבמצוקה - הרי הוא ימשיך לשלוח את אות המצוקה כל עוד הוא לא קיבל Ack...

בוב טוען שלכן המחלקה SignalManager צריכה להיות Active Object. שכן, אם הגיעו מספר אותות מצוקה במקביל ועבור כל אחד הפעלנו את ()sendAck, אז היות ו ()sendAck הופכת להיות אסינכרונית אז לא תהיה בעיה לשלוח לשאר הרובוטים Ack, גם כאשר אחד מהם לא מצליח לקבל את האות.

אליס טוענת שבוב טועה, והמחלקה SignalManager צריכה לעשות שימוש ב Thread Pool.

נמקו מדוע האחד צודק והשני טועה.

תשובה:

אליס צודקת, ב Thread pool נפתח ת'רד אחד לכל משימה. גם אם אחד מהם נתקע בלולאה, זה לא מפריע לת'רד האחר להמשיך לעסוק במשימתו. מה שכן, יש צורך להתאים את מספר הת'רדים למספר הרובוטים שייתכנו במצוקה. (8 נק')

בוב טועה, Active object יוצר הפרדה בין הקריאה למתודה לבין הביצוע שלה – במובן הזה הוא יוצר אסינכרוניות, אך רק במובן זה, מפני שכל בקשה (קריאה למתודה) מכניסה משימה לתור, ורק ת'רד אחד מטפל בבקשות אלה לפי הסדר, אחת אחרי השנייה. כך שאם כעת נשלפה מהתור משימה שנתקעה בלולאה, היא תקעה את ביצוע שאר המשימות... (8 נק')

נשים לב שבוב אמר שהמחלקה SignalManager צריכה להיות Active Object ולא כל משימה צריכה להיות Active object.

שאלה 2: (12 נק')

הקיפו בעיגול את התשובות הנכונות:

א. static design עלול במקרים מסוימים להוביל לקוד כפול
ב. נעדיף קוד שמתאפיין ב high coupling וב low cohesion
ג. ב REST נעשה שימוש ב SOAP אך לא ב WSDL
ד. Memcahed היא טכנולוגיה המאפשרת linear scalability.

מיומנות עיצוב קוד (Design) וכתיבת קוד

שאלה 3: (20 נק') נתון לנו הממשק SignalReader המגדיר את המתודה read() שמחזירה double. בנוסף, נתונות לנו המחלקות RFSignalReader, HFSignalReader, UHFSignalReader שמימשו את הממשק, כל אחת עבור סוג אחר של אות.

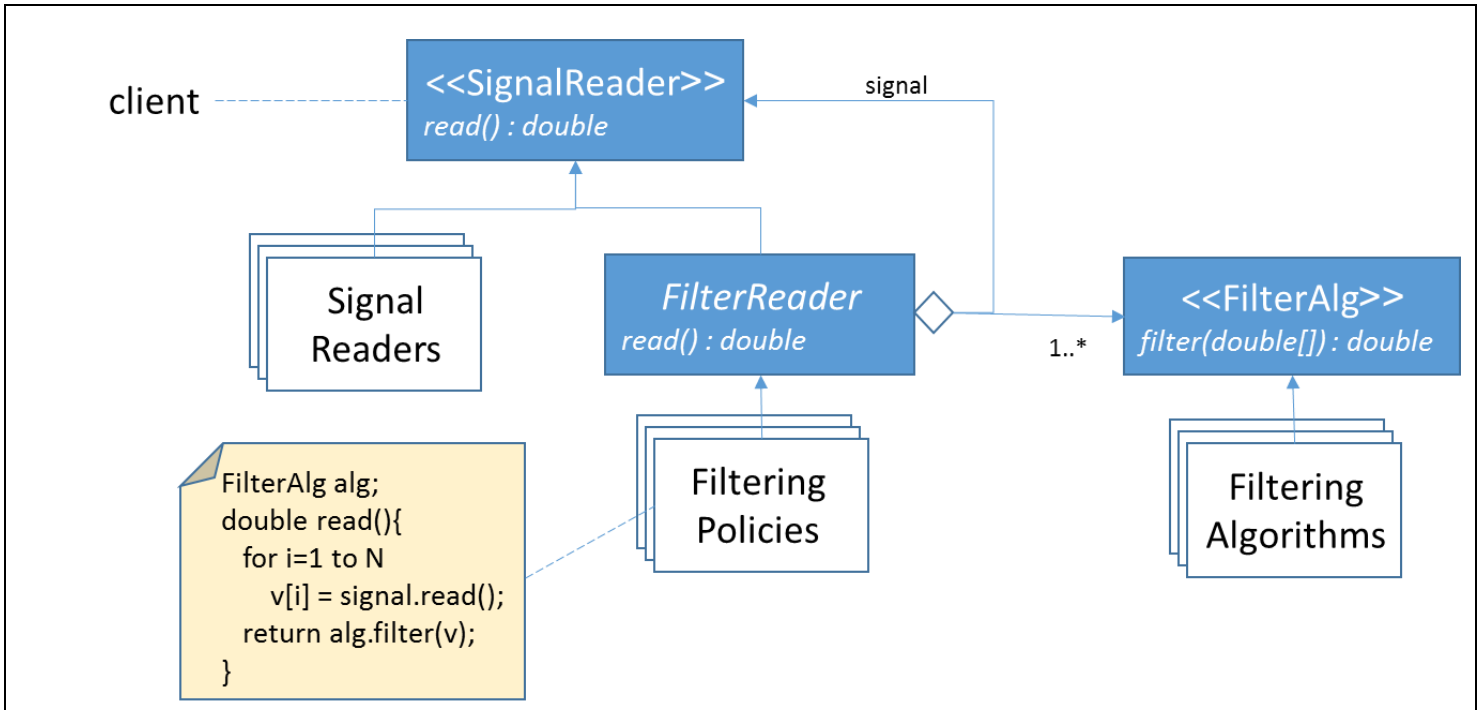
אולם, האותות המוחזרים ע"י מחלקות אלו הינם "רועשים". כלומר, בכל דגימה (הפעלה של read) הערך שמוחזר לנו נע מסביב לערך האמתי. לדוג' אם הערך האמתי הוא 88 אז חוזרים לנו ערכים מסביב ל 88 כמו 88.2, 88.3, 87.9 וכו'.

רעשים אלו צריך לסנן ולהחזיר את הערך המשוער כאמתי (לדוג' 88). המחלקות MovingAverage ו KalmanFilter מימשו כל אחת אלגוריתם שונה לסינון רעשים במתודה double filter(double[] v).

עליכם **לשרטט** תרשים מחלקות (class diagram) **ב UML**, המציג עיצוב שמאפשר לסנן את הרעשים המוחזרים ע"י RFSignalReader ודומיה. בפרט:

- המחלקה FilterReader תהיה אחראית לסינון רעשים
- הקליינט צריך להכיר רק את SignalReader (4 נק') לכן האימפלמנטציה של SignalReader
  - כלומר אין זה משנה לו מהו הסוג הספציפי של SignalReader, ועדיין נרצה לאפשר לו לקרוא אותות מסוננים.
- נרצה יכולת לסנן כל סוג של אות (4 נק') לכן ההכלה של SignalReader
- ובאמצעות כל אלגוריתם סינון (6 נק') לכן ההכלה של FilterAlg
- נרצה שהעיצוב ישמור על עיקרון ה open / close (6 נק') לכן ההרחבה של FilterReader

תשובה: (על השרטוט להכיל את כל המחלקות הרלוונטיות + הערות ע"פ הצורך)



תמיד נוכל להוסיף קורא אותות חדש, אלג' חדש ומדיניות חדשה, מבלי לפגוע בקוד קיים.
המחלקה FilterReader היא סוג של SignalReader וגם מכילה אובייקט כזה, יהיה אשר יהיה, שבאמצעותו היא מבצעת את הקריאה.
המחלקה FilterReader מכילה גם FilterAlg ולכן גם אינה תלויה באלג' ספציפי. ניתן לשים רשימה של אלג' עבור מדיניות המורכבת מהפעלה של כמה אלג' שונים.

טעויות נפוצות:

- סימון הפוך של הכלה (-2)
- פגיעה בעקרון open \ cose ע"י
  - חוסר הרחבה של FilterReader (-3)
  - האלגוריתמים מרחיבים את FilterReader (במקום להיות מוכלים בתוכו), וממשים מתודה אבסטרקטית של FilterReader. זה יוצר חוסר הפרדה בין המדיניות (למשל לסנן 100 דגימות) לבין אלג' הסינון (למשל קלמן). זהו static design שיגרום לכך שכל מדיניות נצטרך להרחיב ע"י כל אלג' סינון. טעות כזו מהווה 4 נק'.

שאלה 4: ממשו בקוד (באיזו שפה מונחית עצמים שתרצו) את המחלקה FilterReader ע"פ העיצוב שלכם משאלה 3, כך שתפעל באופן הבא: בהינתן מקור הקלט הרועש s, ובהינתן אלגוריתם הסינון a, עבור כל בקשת read אחת מ FilterReader נבצע 100 בקשות read מ s, נזין אותן ל a, ונחזיר את הערך האמתי המשוער. (16 נק')

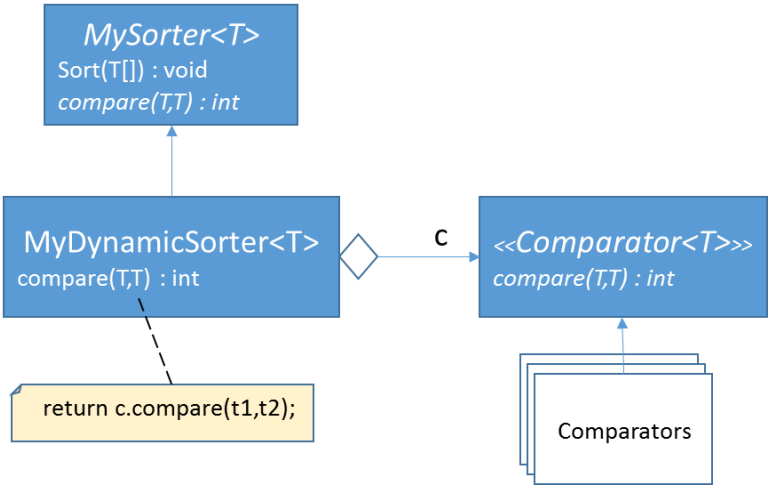
תשובה:

```
public abstract class FilterReader implements SignalReader{ // התאמה לעיצוב 8 נק'

        SignalReader signal;

        FilterAlg alg; // this is just one, you can change with a list of algorithms

        public FilterReader(SignalReader signal, FilterAlg alg){

                this.signal=signal;

                this.alg=alg;

        }

}

public class MyFilterReader extends FilterReader{ // מימוש מחלקות נכון 4 נק'

        public MyFilterReader(SignalReader signal, FilterAlg alg){

                super(signal,alg);

        }

        public double read(){ // מימוש המדיניות 4 נק'

                double[] v=new double[100];

                for(int i=0;i<100;i++)

                        v[i]=signal.read();

                return alg.filter(v);

        }

}
```

שאלה 5: (20 נקודות)

נתונה המחלקה MySorter<T> (כקוד סגור שלא ניתן לשינוי) המאפשרת מיון של אובייקטים מסוג פרמטרי T במתודה sort(T[] arr). כדי לקבוע את הקריטריון ההשוואה בין כל שני T-ים המתודה sort מפעילה (בעת צורך) את המתודה האבסטרקטית abstract int compare(T t1, T t2). לצערנו זהו static design.

עליכם **לשרטט** תרשים מחלקות (class diagram) **ב UML**, שעושה שימוש חכם ב MySorter<T> (5 נק') אך מציג dynamic design עבור בעיית המיון (15 נק').



שאלה 6: ממשו בקוד (באיזו שפה מונחית עצמים שתרצו) את העיצוב של שאלה 5 והדגימו במתודת main כיצד נמיין מערך של אובייקטים מסוג Student ע"פ הגילאים שלהם (ניתן להניח כי קיימת מתודת getAge במחלקה Student). (16 נק') תשובה 6:

```
public interface Comparator<T>{ int compare(T t1, T t2); } // התאמה לעיצוב 8 נק'

public class MyDynamicSorter<T> extends MySorter<T>{ // מימוש מחלקות נכון 4 נק'

        Comparator<T> c;

        public MyDynamicSorter(Comparator<T> c){ this.c=c;}

        public int compare(T t1, T t2) { return c.compare(t1,t2); }

}

// in the main()

new MyDynamicSorter<Student>(  // מימוש נכון 4 נק'

        new Comparator<Student>(){

                int compare(Student s1, Student s2){

                        return s1.getAge()-s2.getAge();

                }

        }

).sort(students);
```