הנחיות לפתרון תרגילי הבית

- על הקוד המוגש להיות מתועד היטב ועליו לכלול:
  - מפרט, כפי שהודגם בתרגול.
  - תיעוד של כל מחלקה ומתודה ושל קטעי קוד רלוונטיים.
  - במידת הצורך, יש להוסיף תיעוד חיצוני.
- יש להפעיל את הכלי Javadoc כדי ליצור קבצי תיעוד בפורמט HTML ולצרף אותם לפתרון הממוחשב המוגש. כדי לגרום לקובצי ה-HTML להכיל את פסקאות המפרט שבהן אנו משתמשים, יש לציין זאת במפורש. ב-Eclipse, ניתן לבצע פעולה זו באופן הבא: 1. לבחור Export מתפריט File, לבחור Java->Javadoc וללחוץ על כפתור Next, 2. לבחור עבור Javadoc command את הקובץ javadoc.exe מתוך התיקייה bin הנמצאת בתיקייה שבה מותקן ה-Java SDK, 3. לבחור את הקבצים שלהם מעוניינים ליצור תיעוד וללחוץ פעמיים על כפתור Next, 4. להקיש ב-Extra Javadoc options את השורה הבאה וללחוץ על כפתור Finish:

-tag requires:a:"Requires:" -tag modifies:a:"Modifies:" -tag effects:a:"Effects:"

- התנהגות ברירת המחדל של פעולות assert היא disabled (הבדיקות לא מתבצעות). כדי לאפשר את הידור וביצוע פעולות assert, יש לבצע ב-Eclipse את הפעולות הבאות: 1. מתפריט Run לבחור Debug Configurations , 2. בחלון שנפתח, לעבור ללשונית Arguments, 3. בתיבת הטקסט VM arguments לכתוב ea–, 4. ללחוץ על כפתור Debug.

הנחיות להגשת תרגילי בית

- תרגילי הבית הם חובה.
- ההגשה בזוגות בלבד.
- עם סיום פתירת התרגיל, יש להגיש את הקבצים הבאים
  - קבצי Java בנפרד
  - זיפ עם קבצי Javadoc
  - פתרון לשאלות ה"יבשות" בקובץ Word או PDF. על הקובץ להכיל את שמות ומספרי תעודות הזהות **של שני הסטודנטים** המגישים.
- הגשת התרגיל היא **אלקטרונית בלבד**, דרך אתר הקורס **ע"י אחד מבני הזוג בלבד**.

- תרגיל שיוגש באיחור וללא אישור מתאים (כגון, אישור מילואים), יורד ממנו ציון באופן אוטומטי לפי חישוב של 2 נקודות לכל יום איחור.
- על התוכנית לעבור קומפילציה. על תכנית שלא עוברת קומפילציה יורדו 30 נקודות.

מועד ההגשה:
יום ג', 26/1/21

המטרות של תרגיל בית זה הן להתנסות בתחומים הבאים:

- תכן של תוכנה תוך שימוש בעקרונות שנלמדו בקורס.
- הבנה ושימוש ב-design patterns.

שאלה 1 (10 נקודות)

נתון חלק מהמפרט של ארבע המחלקות והממשק הבאים מתוך החבילות java.awt ו-javax.swing הסטנדרטיות של שפת Java:

```
/**
/**
* Defines the interface for classes that know how to lay out Containers.
*/
public interface LayoutManager {
//...
}
```

```
/**
* A flow layout arranges components in a left-to-right flow, much like lines
* of text in a paragraph. Flow layouts are typically used to arrange buttons
* in a panel. It will arrange buttons left to right until no more buttons fit on
* the same line. Each line is centered…
*/
public class FlowLayout implements LayoutManager {
//…
}
```

```
/**
* The GridLayout class is a layout manager that lays out a container's
* components in a rectangular grid. The container is divided into equal-sized
* rectangles, and one component is placed in each rectangle…
*/
public class GridLayout implements LayoutManager {
//...
}
```

```
/**
* A generic AWT container object is a component (an object having a
* graphical representation that can be displayed on the screen and that can
* interact with the user) that can contain other AWT components.
*/
public class Container extends Component {
 /**
 * Sets the layout manager for this container.
 */
 public void setLayout(LayoutManager manager);
 /**
 * Validates this container and all of its subcomponents. The validate
 * method is used to cause a container to lay out its subcomponents
 * again. It should be invoked when this container's subcomponents are
 * modified (added to or removed from the container, or layout-related
 * information changed) after the container has been displayed.
 */
public void validate();
//...
}
```

א.
וריאציה על איזה design pattern ממומשת כאן? איזו בעיה בא design pattern זה לפתור?

ב.
ציירו תרשים מחלקות (class diagram) המכיל את המחלקות והממשקים הנ"ל ואת אלה המוזכרים בקוד. על התרשים להכיל את הקשרים בין המחלקות וממשקים אלו. הסבר בעזרת התרשים את אופן פעולת הdesign pattern.

להגשה "יבשה": תשובות לשאלות הנ"ל.

שאלה 2 (40 נקודות)

בשאלה זו נתכנן ונממש אבטיפוס של מערכת צ'ט למספר משתמשים. עבור כל משתמש במערכת יהיה ממשק משתמש שיורכב מתיבת קלט, בה יוכל להקיש שורת טקסט אחת, ומתיבת שיחה, בה יוכל לצפות בטקסט השיחה שלו עם המשתמשים אחרים. אם נניח כי משתמש בשם Student1 מבצע צ'ט עם שני משתמשים נוספים בשם Student2 ו-Student3, הטקסט בתיבת השיחה עשוי להראות כך:

```
Student1: Hi everyone.
Student2: Hi.
Student3: Good morning, how are you today?
Student2: Great.
Student1: Great here too.
```

בתיבה הנ"ל, השורות בשחור הוקשו ע"י Student1 והשורות בירוק ע"י המשתמשים האחרים. שמות המשתמשים הוספו ע"י המערכת.

כל אחד מהמשתמשים יוכל לכתוב טקסט בתיבת הקלט שלו, שיופיע בתיבת השיחה של כל המשתמשים (כולל של עצמו) לאחר לחיצה על מקש Enter. ניתן יהיה לכוון את המערכת כך שכל הטקסט בתיבות השיחה יוכל להופיע באחת מהצורות הבאות:

- טקסט בגופן רגיל.
- טקסט בגופן רגיל מודגש (bold).
- טקסט בגופן שונה.

עליכם לתכנן ולממש בקוד את מערכת הצ'ט הנ"ל תוך שימוש ב-design patterns הבאים: Observer, Strategy. יש להשתמש ב-Swing כדי להציג את ממשק המשתמש. לשם פשטות, ממשק המשתמש של כל משתמש יהיה JPanel וכל ה-JPanels יאוגדו ב-JFrame אחד.

הנחיה: לשם מימוש הממשק הגרפי ניתן להשתמש במחלקות JLabel ו-JTextPane מתוך Swing. עוד מידע ניתן למצוא ב:
http://docs.oracle.com/javase/tutorial/uiswing/components/label.html
http://docs.oracle.com/javase/tutorial/uiswing/components/editorpane.html

להגשה ממוחשבת: מימוש מערכת הצ'ט כולל מפרט כפי שנלמד בקורס ועם representation invariant ו-abstraction function, כולל קריאות מתאימות ל-checkRep().
להגשה יבשה: תיעוד חיצוני המסביר את התכן והמימוש.

שאלה 3 (25 נקודות)

בשאלה זו נעסוק בתוכנה שתייצג ביטויים אריתמטיים. הפעולות האריתמטיות שיתמכו הן חיבור, חיסור, כפל, חילוק ופעולת מינוס אונרי. מחלקת הבסיס בה נשתמש תקרא Expression והיא תספק, בין השאר, את שתי הפעולות הבאות:

- eval() – חישוב הערך המספרי של הביטוי.
- toString() – החזרת הביטוי כמחרוזת.

עליכם לתכנן ולכתוב קוד המשתמש ב-design pattern שנלמד בקורס כדי לייצג ביטויים אריתמטיים. הקוד יאפשר, למשל, בנייה של ביטוי באופן הבא:

```
Expression e =
   new Multiplication(
      new Addition(
         Double.valueOf(2.5),
         Double.valueOf(3.5)),
      new UnaryMinus(
         Integer.valueOf(5)));
System.out.println(e.eval());        // should print out -30.0
System.out.println(e.toString());    // should print out ((2.5 + 3.5) * (-(5)))
```

הנחיה: קיימות בחבילות הסטנדרטיות של Java מחלקות בשם Double, Integer וכד', שהן צאצאים של המחלקה Number. עם זאת, לשם פשטות, ניתן לא להשתמש במחלקות אלה במימוש התרגיל.

להגשה ממוחשבת: מימוש בקוד עבור התוכנה לייצוג ביטויים אריתמטיים כולל מפרט כפי שנלמד בקורס ועם representation invariant ו-abstraction function, כולל קריאות מתאימות ל-checkRep(). בנוסף, דוגמת הרצה של הקוד.
להגשה "יבשה": תיעוד חיצוני של הקוד, כולל הסבר על ה-design pattern שנבחר באופן כללי ובאופן פרטני לפתרון הבעיה הנתונה.

שאלה 4 (5 נקודות)

בתרגיל בית מס' 3 ביצעתם תכן עבור תוכנה לניהול מסעדה. תנו דוגמה לביטוי של עקרון Creator מ-GRASP בתכן שלכם ותנו דוגמה לביטוי של עקרון Information Expert בתכן שלכם. בתשובתכם **עליכם לפרט ולצרף תרשימים** מתשובתכם לתרגיל 3.

להגשה "יבשה": תשובות לשאלות הנ"ל, כולל תרשימים כנדרש.

שאלה 5 (20 נקודות)

נתונות המחלקות הבאות (ללא מימוש לשם קיצור):

```
abstract class Soup {}
class VegetableSoup extends Soup {}
class FishSoup extends Soup {}
abstract class ChickenSoup extends Soup {}
```

המחלקה המופשטת Soup מייצגת מרק ושלוש המחלקות האחרות מייצגות סוגים שונים של מרק – מרק ירקות, מרק דגים ומרק עוף, בהתאמה.
מרק עוף הוא מומחיות מקומית, ולכן הוא נעשה באופן שונה במקומות שונים בעולם.
המחלקה המופשטת SoupMaker יכולה לשמש, למשל, במסעדה, והיא מאפשרת ליצור מרקים מסוגים שונים.

```
abstract class SoupMaker {
	String location;
	public String getLocation() {
		return location;
	}
	public VegetableSoup makeVegetableSoup() {
		return new VegetableSoup();
	}
	public FishSoup makeFishSoup() {
		return new FishSoup();
	}
	public abstract ChickenSoup makeChickenSoup() ;
}
```

{למחלקה זו קיימים שני מימושים שונים, עבור הכנת מרקים בישראל ובאירופה:

```
class IsraeliSoupMaker extends SoupMaker {
	public IsraeliSoupMaker() {
		location = "Israel";
	}
	public ChickenSoup makeChickenSoup() {
		return new IsraeliChickenSoup();
	}
}

class IsraeliChickenSoup extends ChickenSoup {
	public IsraeliChickenSoup() {
		// Create an Israeli chicken soup …
	}
}

class EuropeanSoupMaker extends SoupMaker {
	public EuropeanSoupMaker() {
		location = "Europe";
	}
	public ChickenSoup makeChickenSoup() {
		return new EuropeanChickenSoup();
	}
}

class EuropeanChickenSoup extends ChickenSoup {
	public EuropeanChickenSoup() {
		// Create a European chicken soup …
	}
}
```

דוגמה למחלקת לקוח המשתמשת במחלקות הנ"ל:

```
import java.util.Calendar;

class TestSoupMaker {
	public static Soup makeSoupOfTheDay(SoupMaker soupMaker) {
		Calendar todayCalendar = Calendar.getInstance();
		int dayOfWeek = todayCalendar.get(Calendar.DAY_OF_WEEK);
		if (dayOfWeek == Calendar.MONDAY)
			return soupMaker.makeVegetableSoup();
		else if (dayOfWeek == Calendar.WEDNESDAY)
			return soupMaker.makeFishSoup();
		else
			return soupMaker.makeChickenSoup();
	}

	public static void main(String[] args) {
		SoupMaker soupMaker = new EuropeanSoupMaker();
		Soup soupOfTheDay = makeSoupOfTheDay(soupMaker);
		// eat first soup
		soupMaker = new IsraeliSoupMaker();
		soupOfTheDay = makeSoupOfTheDay(soupMaker);
		// eat second soup
	}
```

א. איזה design patternממומש כאן? איזו בעיית תכן בא design patternזה לפתור? הסבר את התשובה תוך רישום תרשים מחלקות מתאים של המחלקות בשאלה.

ב. התבונן בשורה הראשונה של המתודה makeSoupOfTheDay (במחלקה TestSoupMaker).
איזה design pattern מממשת המחלקה java.util.Calendar? הסבר.

עבודה נעימה!