תכנות מתקדם בשפת Java
בית הספר למדעי המחשב
אוניברסיטת תל אביב

תרגיל מספר 2 – Interfaces and Generics

שאלה מספר 1

בשאלה זו נגדיר את המנשק מחסנית (`IStack<T>`) ונממש שתי גרסאות שלו, `StackImpl1` ו- `StackImpl2`. לקוח של המחסנית ישתמש במפעל (Factory Design Pattern) כדי למזער את התלות שלו במימוש המחסנית לנקודה אחת בתוכנית. בחלק השני של התרגיל נגדיר את המחלקה מקרר (`Fridge`) אשר מהווה בבסיסה Stack של מוצרים (`Product`). נדגים שימוש במחלקה על ידי הגדרת כמה מוצרים (מטיפוסים שונים) והוספתם למקרר. למוצרים השונים תכונות משותפות אך גם תכונות ייחודיות לכל מוצר.

`IStack<T>`

זהו מנשק המתאר מבנה נתונים בשיטת LIFO (מחסנית) אשר מחזיק נתונים מטיפוס כללי (Generic). על המנשק לייצא את השרותים הבאים:

- `isEmpty` – שאילתה המתארת האם מבנה הנתונים ריק
- `isFull` – שאילתה המתארת האם מבנה הנתונים מלא
- `push` – הוספת איבר למבנה הנתונים. ההוספה מוגדרת רק אם מבנה הנתונים אינו מלא
- `top` – מחזירה הפנייה לאיבר העליון במחסנית. הפעולה מוגדרת רק אם מבנה הנתונים אינו ריק
- `pop` – מסירה את האיבר העליון מהמחסנית. הפעולה מוגדרת רק אם מבנה הנתונים אינו ריק

הגדירו מנשק המתאר 5 פעולות אלו. שימו לב לחתימת המתודות והקפידו על ההפרדה בין query ובין command.
תארו את המנשק בשיטת Design by Contract (ראו הסבר בנספח לתרגיל). השתדלו לבטא את טענותיכם בצורה פורמלית ככל הניתן (ביטויים בולאניים ב java). שימו לב – לא תמיד ניתן לתאר מחלקה או מנשק במלואם בעזרת Design by Contract, במקרה זה יש להוסיף תאור מילולי של התנאי. אם לצורך החוזה (או חוזה המימוש) יש להוסיף שאילתות חדשות (שאינן סותרות את מהות המחסנית) ניתן להוסיף אותן **למחלקות המממשות**.

`StackImpl1` ו- `StackImpl2`

כל אחת משתי המחלקות תממש את המנשק בצורה שונה תוך שימוש במחלקות ספרייה של Java. (מחלקות המממשות מנשקים מתוך java.util). מימוש המתודות יבצע האצלה (delegation) לפעולות המוגדרות על אותם מבני נתונים. כמובן שבחירה נכונה של מבני הנתונים בייצוג הפנימי, תשפיע על קלות המימוש.

המחלקה מקרר (שתוצג למטה) היא לקוחה של מנשק המחסנית ואינה מעוניינת להיצמד למימוש מסוים שלו. מפעל מחסניות (Factory Design Pattern) הוא אשר יהיה אחראי לייצר את המחסנית הממשית.

`StackFactory`
היא מחלקת המפעל שמפעילה לוגיקה (אולי טריויאלית) לייצור מחסניות חדשות. המחלקה תממש את המתודה:
`IStack<T> createStack()` אשר תחזיר את עצם מאחד הטיפוסים המממשים את המנשק.
יש לכתוב את המחלקה `StackFactory` כ-Singleton כלומר יש לוודא שלא ניתן יהיה לייצר ממנה יותר ממופע אחד במהלך התוכנית. לדוגמא: אם בתוכנית יהיו שני מקררים שניהם ישתמשו באותו מפעל.
הערה: מכיוון ש `IStack` הוא מנשק כללי (`Generic`) ה-Singleton של מפעל עבורו מקבל משמעות שונה קצת. במקרה זה אפשר לחשוב על Singleton per Type , כלומר מפעל אחד עבור כל מחסנית עם טיפוס כלשהו. בתרגיל זה נסתפק ביצירת Singleton עבור מפעל למחסניות של `Product` (ראו למטה).

`Fridge`
מחלקה זו היא לקוחה של המנשק `IStack<T>`. המקרר ימומש ע"י **מערך של מחסניות** של איברים המממשים את המנשק `Product` (המייצג טורים של מוצרים שונים במקרר).

כאשר קונים מוצרים בצרכנייה ורוצים להכניסם למקרר, אזי המוצרים שכבר נמצאים בקידמת המקרר לא יכולים להידחף אחורה (גם אם יש מקום!). יש להוציא אותם החוצה (אל השולחן שבחוץ!) להכניס את המוצרים החדשים יותר תחילה, ורק אז להחזירם חזרה אל הטור שלהם.

אם בתהליך התגלה מוצר שפקע תוקפו, מוצר זה לא יוחזר למקרר.

המחלקה `Fridge` תממש לפחות את המתודות הבאות:

- `Product pickProduct(String name)`
- `void loadFridge(Collection<Product> supply)`
- `String toString()` – להדפסת המלאי הנוכחי

`Product`
המנשק יכיל לפחות את השרותים הבאים:

- `getExpirationDate`
- `getName`
- `getPrice`

למוצרים השונים תכונות משותפות אך גם תכונות ייחודיות לכל מוצר. שימו לב להבדל שבין יצירת כמה מחלקות המממשות אותו מנשק לבין יצירת עצמים שונים ממחלקה מסוימת. למחלקות שונות המממשות את המנשק `Product` אמורות להיות תכונות ייחודיות להן, אחרת ניתן היה ליצור מופעים של מחלקות קיימות עם ערכים שונים בשדותיהם.

הפתרון צריך לכלול גם מחלקת לקוח (`Makolet`) המציגה שימוש לא טריויאלי בכל המחלקות והמנשקים שהוגדרו לעיל.

שאלה מספר 2

נתון פסאודו-קוד למבנה הנתונים `Node` המייצג קודקוד בעץ בינארי, המכיל ערכים מטיפוס `T`:

```
class Node {
     Node left;
     Node right;
     Node parent;
     T val;
     // maybe more methods and fields...
}
```

בני מחלקה בשם `Tree` המממשת אוסף (collection) של נתונים המוחזקים בעץ בינארי. על המחלקה לתמוך בשרותי אצן (איטרטור) סטנדרטיים (ע"י הגדרת מחלקה פנימית מהטיפוס המתאים). אין צורך לתמוך בכמה שיטות לסריקה של העץ. הגדירי אצן לפי שיטת הסריקה שבחרת (למשל LDR). יש צורך לממש הן את המתודות של `Tree` והן את המנשק של האצן.

<u>הדרכה</u>: לב התרגיל הוא מימוש המתודה `next()` של האצן. מתודה זו מבטאת את המעבר מרקורסיה ללולאה. בלולאה יש צורך "לזכור" באיזו נקודה אנו נמצאים בביצוע המעבר על העץ – דבר זה מתאפשר ע"י הוספת משתני עזר לאצן שיעזרו לו לזכור איפה הוא הפסיק.

ממשי את הפונקציות הנדרשות כדי שהקוד הבא ידפיס את תוכנו של העץ המוצבע ע"י הארגומנט `t`:

```
public void printTree(Tree<someType> t) {
     for (someType elem : t)
          System.out.print(elem + " ");
     System.out.println("") ;
}
```

ממשי את הפונקציות הנדרשות כדי שהקוד:

```
Tree<someType> t = ...
system.out.prinln(t);
```

ידפיס את תוכנו של `t`. המימוש יכול להשתמש בתכונות שמומשו קודם.

יש לספק מחלקה בשם `TestTree` המדגימה את פעולת המחלקות

<u>מה להגיש?</u>

JAR בשם ex2.jar המכיל את החבילות והקבצים הבאים:

- il.ac.tau.cs.advJava.ex2.stacks
  - IStack.java
  - StackImpl1.java
  - StackImpl2.java
  - Fridge.java
  - Product.java
  - StackFactory.java
  - Makolet.java – תכיל פונקציית main
- il.ac.tau.cs.advJava.ex2.iterators
  - Tree.java
  - Node.java
  - TestTree.java – תכיל פונקציית main
- תיקיית doc/ שתכיל את הפלט של javadoc (חובה לכל המחלקות)

יש לעבוד בסביבת הפיתוח בתצורת תיקיות אשר תתחזק שתי היררכיות מקבילות של תיקיות:

- src/ (לקובצי java)

ו-

- classes/ (לקבצים מקומפלים)

כמו כן, יש לספק תדפיסים של כל קובצי המקור.
אין צורך להדפיס את פלט ה javadoc.

נספח: ניסוח טענות בשיטת Design by Contract

1. תנאי קדם:

@pre boolean , String

על הביטוי הבולאני להשתערך ל TRUE לפני הקריאה לפונקציה. תנאי זה לא ייבדק בגוף המתודה והוא על אחריות הלקוח בלבד.המחרוזת תכיל תאור מילולי של התנאי

2. תנאי אחר (תנאי בתר):

@post boolean , String

על הביטוי הבולאני להשתערך ל TRUE בסוף ריצת הפונקציה. תנאי זה לא ייבדק ע"י הלקוח לאחר החזרה מהפונקציה והוא על אחריות הספק בלבד. המחרוזת תכיל תאור מילולי של התנאי

$prev(*expression*)

תנאי אחר בולאנים עשויים להכיל ביטוי $prev המבטא את ערכו של *expression* לפני תחילת ריצת המתודה. במהלך הריצה עשוי ערכו של הביטוי להשתנות וכך נוכל להשוות בין הערך החדש והישן לדוגמא:

@post $prev(balance()) == balance() – amount, "reduce balance by amount"

$ret

תנאי אחר בולאנים עשויים להכיל ביטוי $ret המבטא את הערך המוחזר של הפונקציה.

3. שמורת המחלקה (משתמר):

@inv boolean , String

תנאי זה מתייחס למחלקות ולא למתודות. על הביטוי הבולאני להשתערך ל TRUE תמיד, כלומר לפני ואחרי ריצת כל מתודה ציבורית של המחלקה וכן לאחר סיום ריצת הבנאי. תנאי זה לא ייבדק ע"י הלקוח לפני קריאה למתודות של המחלקה או אחרי החזרה מהן והוא על אחריות הספק בלבד. המחרוזת תכיל תאור מילולי של התנאי.

4. שמורת מימוש (שמורת ייצוג):

@imp_inv boolean , String

ניתן להשתמש במנגנון ה"עיצוב לפי חוזה" גם לצורכי פנים, כלומר לתיאור הקוד שלא כלפי הלקוח, מעין כלי נוסף לתיעוד. במקרים אלו יכול התנאי הבולאני להכיל גם רכיבים שאינם ציבוריים (למשל שדות private). על הביטוי הבולאני להשתערך ל TRUE תמיד, כלומר לפני ואחרי ריצת כל מתודה ציבורית של המחלקה וכן לאחר סיום ריצת הבנאי. תנאי זה לא ייבדק ע"י הלקוח מכיוון שהלקוח כלל לא חשוף אליו. המחרוזת תכיל תאור מילולי של התנאי.

5. תנאי אחר לצורכי מימוש:

@imp_post boolean , String

ניתן להשתמש במנגנון ה"עיצוב לפי חוזה" גם לצורכי פנים, כלומר לתיאור הקוד שלא כלפי הלקוח, מעין כלי נוסף לתיעוד. במקרים אלו יכול התנאי הבולאני להכיל גם רכיבים שאינם ציבוריים (למשל שדות private). על הביטוי הבולאני להשתערך ל TRUE בסוף ריצת הפונקציה. תנאי זה לא ייבדק ע"י הלקוח מכיוון שהלקוח כלל לא חשוף אליו. המחרוזת תכיל תאור מילולי של התנאי.