תוכנה 1 – אביב תשע"ט

תרגיל מספר 7

מנשקים Interfaces

הנחיות כלליות:

קראו בעיון את קובץ נהלי הגשת התרגילים אשר נמצא באתר הקורס.

- הגשת התרגיל תעשה במערכת ה-moodle בלבד (http://moodle.tau.ac.il/).
- יש להגיש קובץ zip יחיד הנושא את שם המשתמש ומספר התרגיל (לדוגמא, עבור המשתמש **aviv9** יקרא הקובץ **aviv9**_hw7.zip). קובץ ה-zip יכיל:
  א. קובץ פרטים אישיים בשם details.txt המכיל את שמכם ומספר ת.ז.
  ב. את תיקיית **src** ובתוכה היררכיית התיקיות כפי שקיבלתם בקובץ הזיפ, כולל קבצי הג'אווה שסופקו לכם (אשר נוסף להם הקוד שלכם). כלומר התיקיה src עצמה תהיה בזיפ כך שהיא לא מוכלת באף תיקיה אחרת (וכל המבנה בתוכה יהיה זהה למבנה שאתם קיבלתם, חוץ מהתוכן של קבצי הג'אווה עצמם כמובן).

- נא לא להשתמש בפקודה System.exit()! היא מחבלת בבדיקות אוטומטיות. אין כל צורך לעשות בה שימוש, כאשר תוכניות יכולות להסתיים ע"י הגעה לסוף מתודת ה-main או על ידי פקודות return.

הערות הגשה:

1. יש להגיש את התרגיל בקובץ זיפ בלבד .
2. יש להגיש קוד שמתקמפל, כולל שורות יבוא המחלקות שהשתמשתם בהן, ולא כולל אף סימן שנכנס בטעות לקובץ אחרי שהתרגיל כבר עבד בהצלחה.
3. יש לוודא כי הקוד מוגש במבנה התיקיות כפי שצוין.
4. שם קובץ הזיפ שתגישו יכלול את שם המשתמש האוניברסיטאי ולא את שמכם הפרטי.

יצירת פרוייקט והגשה: חזרו על ההוראות ממטלה 3 לגבי יצירת פרויקט וייבוא הקבצים. התהליך הוא זהה במטלה זו: יש ליצור פרויקט ג'אווה חדש באקליפס (ולשים לב, למיקום של הworkspace במחשבכם). בעת יש להיכנס לתיקיית הפרויקט במחשב, ולהעתיק לשם את התיקיות ה-src ו-resources מתוך קובץ הזיפ, כך שתיקיית ה-src הקיימת תידרס. תיקיית ה-src הזו היא התיקיה שתצרפו לזיפ בתום כתיבת הקוד. חזרו כעת לאקליפס, ובלחיצה ימנית על הפרוייקט בחרו refresh.

הערות לתרגילים:

- מותר לכתוב מתודות עזר, אך יש לשמור את כולן באותו הקובץ, ולא ליצור עבורן מחלקות חדשות.
- אין לשנות חתימות של המתודות שמופיעות בקבצי הקוד כפי שקיבלתם אותם.
- יש להניח כי הקלט לכל מתודה הוא חוקי, ולא לטפל בקלט לא חוקי, אלא אם נאמר אחרת מפורשות.

חלק א' (35 נק')

בחלק זה נתרגל כתיבת מחלקות המממשות מנשק נתון, בנוסף למימוש מתודות סטטיות ו-default במנשק, כולל מנשקים גנריים. הקוד מופיע בחבילה il.ac.tau.cs.software1.predicate .

נתונים המנשקים: `Predicate ,Action, Product`.

והמחלקות: Book, SmartPhone, ByAuthor, ByPrice, Discount, Upgrade, Store.

עליכם להשלים את הקוד החסר במחלקות ובמנשקים בהתאם למצוין בהנחיות.

אנחנו נממש את המחלקה הגנרית Store. ב-Store יש מלאי של מוצר מסוים שמממש את המנשק product. שני מימושים נתונים של המנשק הזה הם ספר (Book) וטלפון (SmartPhone). המתודה המרכזית ב-Store היא Transform אשר מקבלת מחלקה שממממשת את המנשק Predicate שמייצג קריטריון בוליאני כלשהו עבור מוצר, ומחלקה שממממשת את המנשק Action שמייצג פעולה שאפשר לעשות על מוצר, ומבצעת את ה-Action על כל המוצרים במלאי שעומדים ב-Predicate.

- כל המחלקות והמנשקים בחלק זה ימומשו כחלק מחבילה בשם il.ac.tau.cs.software1.predicate
- מותר (ולעתים הכרחי) להוסיף שדות, אך יש להגדירם **פרטיים** בלבד, וגישה אליהם (אם יש צורך בכך! כברירת מחדל, אין צורך אם לא צוין אחרת מפורשות.) תהיה באמצעות מתודות getter ו- setter ציבוריות, שעליכם להוסיף בעצמכם.
- יחד עם השלד לקוד סופקה לכם המחלקה Tester עם מספר בדיקות בסיסיות לתוכנית. אין צורך להוסיף כל קוד למחלקה הזאת (היא נועדה לשימוש אישי). כמו כן, הטסטר אינו מקיף, כלומר בודקי התרגיל יבצעו בדיקות נוספות מעבר לטסטר.
- בתרגיל נעשה שימוש בסיסי במנשק הגנרי java.util.List<E> (נלמד עליו ועל מבנים גנריים דומים בהרחבה בתרגול). (https://docs.oracle.com/javase/8/docs/api/java/util/List.html)
- מחלקה שימושית שממשקת את List<E> היא java.util.ArrayList<E> כפי שתוכלו לראות בטסטר. (https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html)
- ניתן לעשות איטרציה על List<E> ע"י מבנה ה-for each שראינו כבר בפעולה על מערכים:
  for (E element : myList) {….}
- בהצהרות של טיפוסים גנריים אתם תראו שימוש ב- <T extends Product> במקום פשוט <T>. זה אומר שהטיפוס האקטואלי ש-T מייצג מוכרח לקיים יחס is-a עם Product, שזה כולל כמובן כל מחלקה שממממשת את המנשק הזה. זה נועד לכך שנוכל להפעיל על משתנים מטיפוס T מתודות של Product, אשר עבור ההצהרה הבסיסית <T> הקומפיילר היה אוסר, היות ולא מובטח שלטיפוס האקטואלי, שיכול להיות כל עצם, יש מתודות כאלה. הבנה שטחית של ההצהרה הזאת מספיקה לצורכי התרגיל. בהמשך הקורס, נרחיב את הדיון על תכנות גנרי, ונבחן לעומק את ההצהרות הנ"ל.

1. ראשית, קראו את הקוד במנשק Product וממשו בו את המתודה הסטטית:
   static <T **extends** Product> **double** getTotalPrice(List<T> products)
   מתודה זו מקבלת כקלט רשימת מוצרים, וצריכה להחזיר את סכום המחירים של המוצרים ברשימה.
   לאחר מכן, קראו את הקוד של שני המימושים של המנשק הנ"ל: Book ו-SmartPhone. אין כל צורך להוסיף קוד למחלקות האלה.
2. קראו את הקוד של המנשק הגנרי Predicate המכיל את המתודה היחידה test. כעת, בשתי המחלקות שממממשות את המנשק הזה, ByAuthor ו-ByPrice, יש לממש את הבנאי ואת המתודה test.
   המחלקה ByAuthor מקבלת כקלט לבנאי אות (letter) מטיפוס char שישמש אותה במתודה test, אשר מקבלת כקלט ספר ומחזירה true אם ורק אם האות הראשונה בשמו של המחבר היא אותה ה-letter. ניתן להניח כי letter היא תמיד אות אנגלית קטנה (lowercase), ולכן בבדיקה יש בהתאם להמיר את האות הראשונה בשמו של המחבר ל-lowercase גם כן.
   המחלקה ByPrice מקבלת כקלט לבנאי מחיר (maxPrice), ובמתודה test אשר מקבלת כקלט טלפון, יש להחזיר true אם ורק אם המחיר שלו הוא קטן או שווה ל-maxPrice .

3. קראו את הקוד במנשק הגנרי Action המכיל את המתודה היחידה performAction. כעת, נשלים את הקוד בשתי המחלקות שממשות את המנשק הזה: Discount ו-Upgrade.
   במחלקה Discount יש לממש את הבנאי ואת מתודת ה-performAction . הבנאי מקבלת כקלט אחוז (percentage) בין אפס למאה, כך שבמתודת performAction שמקבלת כקלט ספר, המחיר של הספר ירד ל-percentage מהמחיר המקורי. כלומר, אם, למשל, ספר עולה 100 ו percentage = 25, אז המחיר החדש הוא 25. נדגיש שאפשר להניח שהקלט הוא אכן מספר תקין בין 0 ל-100 ואין צורך להתייחס למקרה אחר.

במחלקה Upgrade יש לממש את מתודת performAction שמקבלת כקלט טלפון, ומבצעת עליו את פעולת השדרוג (יש לכך מתודה מתאימה במחלקה SmartPhone).

4. קראו את הקוד במחלקה הגנרית Store, וממשו את המתודה:
   public String getInventoryDescription()
   מתודה זו תחזיר מחרוזת המתארת את ה-inventory בכך תשרשר את מחרוזות ה-description של כל המוצרים לפי סדר הופעתם ברשימת ה-inventory. שימו לב, כי לכל מוצר כבר קיימת במנשק Product מתודה עם מימוש ברירת מחדל:
   default String getDescription()
   אשר תוכלו להשתמש בה.
5. ממשו במחלקה Store את המתודה:
   public void transform(Predicate<T> pred, Action<T> action)
   אשר מפעילה את action על כל מוצר ב-inventory אשר עומד ב-pred (כלומר שהמתודה test של ה-pred תחזיר עליו true). שימו לב, כי אין לשנות בשום שלב (וגם לא בסעיפים הקודמים) את סדר הפריטים ב-inventory.

<u>חלק ב' (15 נק')</u>

<u>הערה</u>: במידה ומשהו לא ברור או חסר בהוראות שלפניכם, לפני שאתם שואלים, עברו ביסודיות על הטסטר, כולל ההערות - הוא עוזר להבין איך הדברים מתחברים ואיך יראה קלט-פלט של התוכנית.

בתרגיל זה נממש גרסה בסיסית של המחלקה BufferedWriter עליה למדתם במדריך ללימוד עצמי לפעולות קלט/פלט. הקוד ימומש בחבילה il.ac.tau.cs.software1.bufferedIO.

כזכור, העיקרון המנחה של מחלקה זו הוא שהיא עוטפת זרמים אחרים (לרוב FileWriter) ודרכם כותבת מספר קבוע של תוים לקובץ, באופן שקוף למשתמש (ה BufferedReader עובד באופן דומה). בכל פעולת כתיבה דרך ה BufferedWriter, כתיבה לקובץ מתבצעת רק אם ה buffer של ה BufferedWriter מלא.

<u>המחלקה MyBufferedWriter</u>:

מחלקה זו מממשת את המנשק IBufferedWriter (מוגדר עבורכם בחבילה bufferedIO).

בנאי המחלקה מקבל:

- fWriter -אובייקט מטיפוס FileWriter. להזכירכם, ה FileWriter מאפשר כתיבה של מחרוזות/מערכי תוים לקובץ.
- bufferSize - מספר שלם וחיובי (גדול מ-0) – גודל ה buffer.
- הבנאי יאתחל buffer שיכיל את התוים שטרם נכתבו לקובץ.

המתודה write:

- מקבלת מחרוזת str שאמורה להיכתב לקובץ ע"י שימוש ב FileWriter. בכל גישה ל FileWriter עלינו לכתוב מספר תוים ששווה לערכו של bufferSize (שאותחל בבנאי).
- מתודה זו תחליט על מספר הכתיבות לקובץ על סמך bufferSize, תוכן ה buffer הנוכחי וכן אורך המחרוזת str שהתקבלה כפרמטר ל write.

- לדוגמא: אם גודל ה buffer הוא 5 תוים ונרצה לכתוב מחרוזת בעלת 7 תווים, 5 תווים יכתבו לקובץ, ו2 תווים ישמרו ב buffer. בפעולת ה write הבאה, אם נרצה לכתוב 2 תווים, הם יצטרפו ל 2 התווים שכבר היו ב buffer והוא יכיל 4 תווים **שלא** נכתבו לקובץ.

המתודה close:

- יש לסגור את ה FileWriter במתודה זו. בנוסף, במידה וקיימים תווים ב buffer שטרם נכתבו, יש לכתוב אותם בפונקציה זו.

<u>בדקו את עצמכם:</u>

הטסטר של חלק זה הוא מאוד בסיסי ובודק את התוכן הנקרא, אך לא את השימוש ב buffer. חלק מהעבודה שלכם היא לתכנן בדיקה שתוכל לבדוק גם את פעולת ה buffer (כלומר, שאתם משתמשים ב FileWriter רק כאשר ה buffer מצריך זאת, ולא בכל פעולת write של המשתמש).

<u>הערה</u>: אם קיימת איזושהי אי התאמה בין הקוד וההנחיות לבין הקובץ rocky1 שסופק לכם לבדיקה מבחינת רווחים, אנחנו נקבל התאמה לכל אחת מהגירסאות.

<u>הנחיה:</u> השתמשו במחלקה MyFileWriter אשר מימושה מופיע בחבילת התרגיל. מחלקה זו יכולה לסייע לכם לנתר את מספר הכתיבות שנעשו לקובץ מחוץ למימוש של MyBufferWriter. מחלקה בנויה על פי design pattern שנקרא decorator והוא מאוד שימוש בבעיות רבות, כמו גם בתרגיל שלנו. ע"י מעבר על מימוש המחלקה הסיקו כיצד ניתן לשלבה בתסריט הבדיקות שלכם. נזכיר שאתם יכולים לעשות שינויים באופן חופשי בטסטרים, כי הם אף פעם לא נבדקים.

<u>הנחות והנחיות נוספות:</u>

א. הניחו כי הכותב שמתקבל כפרמטר בבנאי אינו null ואינו סגור.

ב. <u>אין לייצר ולהשתמש בשום Stream למעט זה שמתקבל ע"י הבנאי של הכותב</u>. שימו לב שאתם לא מקבלים את שם הקובץ ממנו אתם קוראים כך שכל העבודה מתבצעת ע"י הפעלת ה FileWriter שקיבלתם בבנאי. כמות המידע שתיכתב בכל פעם תלויה בגודל ה buffer שגם הוא מאותחל בבנאי.

ג. השתדלו לצמצם את מספר מחרוזות הביניים הנוצרות בריצה אחת של המתודה write. מימוש טוב לא ייצר יותר משתי מחרוזות ביניים בריצה אחת של write.

ד. מבנה הנתונים של ה buffer נתון לשיקולכם. מי שרוצה לכתוב מימוש כמה שיותר קרוב למימוש האמיתי של ה BufferedWriter יכול לנסות ולממש אותו באמצעות מערך של תוים ([]char).

<u>חלק ג' (50 נק') – כתובת IP</u>

הערה: מותר כרגיל להוסיף מתודות עזר, ומומלץ כמו תמיד שיהיו private.
הערה: שימו לב כי בתרגיל זה (בהתאם להערה נוספת שתופיע בהמשך) יכול להיווצר מצב של שכפול קוד - שבאופן כללי, מחוץ לתרגיל הזה, זה לא מצב רצוי - כתוצאה מכך שנרצה לכתוב מימושים נפרדים שלא "יודעים" אחד על השני. בתרגיל הזה זה מצב רצוי.

המנשק IPAddress המופיע למטה מייצג כתובת של Internet Protocol (IP). דוגמאות לכתובות IP הן:

127.0.0.1

192.168.1.10

כתובת IP, כפי שניתן לראות, מורכבת **מארבעה** חלקים. ערכו של כל אחד מהחלקים הוא מספר שלם בין 0 ל-255. בסעיף זה נממש את המנשק IPAddress על ידי שלושה ייצוגים שונים: הראשון עושה שימוש במחרוזות, השני במערך של מספרים מסוג short ואילו השלישי משתמש ב-int יחיד (הסבר מפורט בהמשך).

א. כתבו **שלוש** מחלקות שונות המממשות את המנשק IPAddress המוגדר למטה:
1. מחלקה בשם IPAddressString, המממשת את המנשק בעזרת ייצוג פנימי של String.
2. מחלקה בשם IPAddressShort, המממשת את המנשק בעזרת ייצוג פנימי של מערך בגודל 4 של short. כל תא במערך יחזיק מספר בתחום 0..255.
3. מחלקה בשם IPAddressInt, ה מממשת את המנשק בעזרת int יחיד.

לכל אחת מהמחלקות יהיה בנאי המתאים לייצוג הפנימי שלה וכמובן כל אחת מהן מממשת את המנשק. ניתן להניח בחוזה שהבנאים מקבלים קלט תקין ליצירת כתובת IP (בהתאם לייצוג הפנימי של כל מחלקה).

```
public interface IPAddress {

	/**
	 * Returns a string representation of the IP address, e.g.
	 * "192.168.0.1"
	 */
	public String toString();

	/**
	 * Compares this IPAddress to the specified object
	 * @param other
	 *            the IPAddress to compare the current against
	 * @return true if both IPAddress objects represent the same
	 * IP address, false otherwise.
	 */
	public boolean equals(IPAddress other);

	/**
	 * Returns one of the four parts of the IP address. The parts
	 * are indexed from left to right. For example, in the IP
	 * address 192.168.0.1 part 0 is 192, part 1 is 168,
	 * part 2 is 0 and part 3 is 1.
	 * (Each part is called an octet as its representation
	 * requires 8 bits.)
	 * @param index
	 *            The index of the IP address part (0, 1, 2 or 3)
	 * @return The value of the specified part.
	 */
	public int getOctet(int index);




	/**
	* There are four classes of private networks
	* (http://en.wikipedia.org/wiki/IPv4#Private_networks)
	* 10.0.0.0 - 10.255.255.255
	* 172.16.0.0 - 172.31.255.255
	* 192.168.0.0 - 192.168.255.255
	* 169.254.0.0 - 169.254.255.255
	*
	* This query returns true if this object is a private network address
	*/
	public boolean isPrivateNetwork();
```

```
}
```

להלן פירוט לגבי אופן מימוש הייצוגים השונים:

1. מחרוזת – יש להשתמש במחרוזת יחידה לצורך ייצוג כתובת ה IP . כל הפעולות יבוצעו בעזרת מחרוזת זו.
2. מערך – כל אחד מחלקי הכתובת (מספר שלם 0-255) יוחזק בתא במערך.
3. int – נשים לב שכל אחד מחלקי כתובת ה-IP הוא מספר שלם בתחום 0-255 (כולל), לפיכך ניתן לייצג אותו בעזרת 8 ביטים. על כן, את ארבעת חלקי הכתובת ניתן לייצג באמצעות 32 ביטים (4 בתים) וזהו בדיוק גודלו של int.

לפיכך, נשתמש ב-int לא כמספר, אלא כרצף בינארי של 32 ביטים. לדוגמא, הכתובת 127.0.0.1 תיוצג ע"י רצף הביטים 00000001**00000000**00000000**01111111**.

החלק הראשון ייוצג ע"י הביטים במקומות 0-7 (משמאל לימין), החלק השני ע"י הביטים 8-15, השלישי ע"י 16-23 והרביעי ע"י ביטים 24-31.

127 <- 01111111 (ביטים 0-7 משמאל לימין)
0 <- 00000000 (ביטים 8-15)
0 <- 00000000 (ביטים 16-23)
1 <- 00000001 (ביטים 24-31)

<u>הנחיה:</u> על מנת להשתמש בייצוג זה, עליכם לדעת איך לחלץ את ערכו של כל בית (8 ביט) מהרצף הבינארי באורך 4 הבתים (32 ביטים) שמרכיב את ה-int. נציע 2 דרכים אפשריות:

1. שימוש באופרטורים על ביטים (<<,>>,~,&,|) להזזה או למיסוך של הביטים ברצף הבינארי כך שיאופסו כל הביטים מלבד אלו השייכים לבית הרצוי.
2. שימוש במחלקה [ByteBuffer]

- צרו אובייקט בגודל 4 בתים ע"י ByteBuffer.allocate(4)
- היעזרו במתודות put(i)/get(i) להצבה ולחילוץ של בית במיקום i.
- שימו לב שהמתודה get(i) תחזיר את הבית באינדקס i באובייקט ה- ByteBuffer שיצרתם כמשתנה מטיפוס byte עם טווח ערכים של (-128..+127).

כדי לבצע המרה של בית b מטיפוס byte למשתנה מטיפוס int עם טווח הערכים 0..255 לו אנו זקוקים, ניתן להשתמש בטריק הבא:

```
int i = (int) (b & 0xFF);
```

הערה חשובה: עליכם לממש את המתודות באופן שונה בכל מחלקה בהתאם לייצוג הפנימי. אין להמיר את הייצוג הפנימי לייצוג אחר לצורך מימוש פעולה (רק לצורך פלט). המחלקות השונות לא "ייעזרו" זו בזו (למשל, אסור להשתמש ב-IPAddressString על מנת לממש את IPAddressInt). **לא משתפים קוד בין מימושים**. אם זה יוצר שכפול קוד, זה בסדר לצרכי התרגיל. ובאופן כללי, אם יש ספק אם משהו נחשב כשיתוף בין מימושים (למשל מתודות עזר), ההנחיה הכללית היא להעדיף ליצור שכפול קוד מאשר כל צורה של שיתוף. וזה בסדר גמור אם יהיו חלקים דומים או זהים בין המימושים.

בנוסף, ממשו את המחלקה IPAddressFactory המגדירה את המתודות הבאות:

```
public class IPAddressFactory {

	public static IPAddress createAddress(String ip) {
		...
	}

	public static IPAddress createAddress(short[] ip) {
		...
	}
```

```
	public static IPAddress createAddress(int ip) {
		...
	}
}
```

כל אחת מהמתודות הסטטיות יוצרת אובייקט מטיפוס IPAddress, כשהאובייקט הקונקרטי נקבע על סמך טיפוס הקלט.

הערה: מחלקה שתפקידה היחיד הוא יצור אובייקטים של מחלקות אחרות נקראת *factory class*. מחלקות אלו מסתירות את פרטי יצור האובייקטים מלקוחות של אובייקטים אלו. השימוש בטכניקה זו נועד להסתיר את המחלקות הקונקרטיות שממשות מנשק.

להלן תכנית המדגימה את השימוש במחלקה IPAddressFactory ובמנשק.

```
public class TestIPAddress {

	public static void main(String[] args) {
		int address1 = -1062731775; // 192.168.0.1
		short[] address2 = { 10, 1, 255, 1 }; // 10.1.255.1

		IPAddress ip1 = IPAddressFactory.createAddress(address1);
		IPAddress ip2 = IPAddressFactory.createAddress(address2);
		IPAddress ip3 = IPAddressFactory.createAddress("127.0.0.1");

		for (int i = 0; i < 4; i++) {
			System.out.println(ip1.getOctet(i));
		}

		System.out.println("equals: " + ip1.equals(ip2));
	}
}
```

מצורפת תכנית בשם **TestIPAddress** המדגימה את השימוש במחלקה IPAddressFactory ובמנשק.

<u>הערה</u>: חשוב ששלושת המימושים לא יכירו אחד את השני - לא יחלקו קוד או יכירו את המימוש הפנימי של האחרים או יסתמכו עליו.

בחלק זה עליכם להגיש את כל הקבצים המצורפים בתיקייה ip. הקובץ TestIPAddress, נועד לבדיקה עצמית בלבד, אותו אתם יכולים להרחיב ולשנות ולבדוק את עצמכם (ניתן להגיש גם אותו, אך הוא לא יבדק).

<u>בהצלחה!</u>