

**בחינה בתוכנה 1**

סמסטר ב' תשפ"א, מועד ב', 12 בספטמבר 2021  
לנה דנקין, אביב ביק, עמית כהן

משך הבחינה שלוש שעות.

סך הניקוד על השאלות בבחינה הוא 105, אך הציון המקסימלי אותו ניתן לקבל הוא 100.

יש להניח, אלא אם צויין אחרת, כי:

- הקוד שמופיע במבחן מתאים לגירסא Java8.
- כל החבילות הדרושות יובאו, ואין צורך לכתוב שורות import בגוף הקוד.
- כל מחלקה שהיא public מופיעה בקובץ Java משלה.
- בכל שאלה, כל המחלקות מופיעות באותה חבילה (package).
- בזמן הבחינה, אתם נדרשים לזהות שגיאות קומפילציה שנוצרות כתוצאה מהפרת עקרונות Java-יים ושימוש לא נכון במחלקות/פונקציות. במידה וישנה טעות הקלדה (סוגר חסר, שימוש באות גדולה שלא לצורך וכו') אין לראות בסיבות אלה גורמים לשגיאות קומפילציה.
- בסוף הבחינה מופיע נספח עם תיעוד של מחלקות שאתם עשויים לעשות בהן שימוש בחלק הפתוח של הבחינה.

בבחינה זו מופיע קוד שבחלקו אינו מתקמפל, אינו רץ או שנוגד את הסטנדרטים של Java כפי שנלמדו בקורס, וזאת מתוך מטרה לבחון ידע והבנה של נושאים מסוימים. אין לראות בקטעי קוד אלה דוגמא לכתובה נכונה ב Java.

**מבנה הבחינה:**

הבחינה מורכבת משני חלקים: חלק פתוח (שתי שאלות על סך 50 נקודות) ושאלות אמריקאיות (11 שאלות, כל אחת שווה 5 נק'). עליכם לענות על הבחינה באופן הבא:

- בשאלות הפתוחות להשלים את הקוד החסר במקומות המסומנים ע"י מסגרת. שימו לב שלא חייבים למלא את כל המסגרות.
- בשאלות האמריקאיות:
  - לסמן את התשובות הנכונות על גבי טופס סימון התשובות שתקבלו בנפרד.
  - לנמק את תשובתכם על גבי טופס הבחינה. הנימוק הוא לא חובה, אך יכול לעזור לכם במקרים של ערעורים או קבלת יותר מתשובה אחת נכונה.

בסוף שאלה 2 ניתן למצוא מסגרת חירום לשימוש במקרה שהמסגרות שמופיעות בגוף השאלות הפתוחות לא מספיקות לכם.

© כל הזכויות שמורות למחברים. מבלי לפגוע באמור לעיל, אין להעתיק, לצלם, להקליט, לשדר, לאחסן במאגר מידע, בכל דרך שהיא, בין מכונית ובין אלקטרונית או בכל דרך אחרת כל חלק שהוא מטופס הבחינה.

בהצלחה!

**שאלה 1 (35 נק' ):**

בשאלה זו נממש מערכת שמנהלת הזמנות של מנות במסעדה. המערכת שלנו תנהל את החלק שאחראי על שליחת ההזמנות שנלקחו ע"י המלצרים למטבח. המערכת תאפשר את הפעולות הבאות: ביצוע הזמנה, ביטול הזמנה, וכן טיפול בהזמנה ע"י שליחתה למטבח.

כל הזמנה מורכבת משני חלקים: שם המזמין וסוג המנה. אנחנו נייצג את כל המנות הקיימות בתפריט המסעדה באמצעות ה enum שנקרא MenuItem. לשם פשטות, הגדרנו בו שלוש מנות.

```
public enum MenuItem{
    SAGIT_SALD, EGG_AVO_BREAD, AMERICANO
}
```

את ההזמנה נייצג ע"י המחלקה Order. נשתמש במחלקה זו על פי דוגמת השימוש:

```
Order o1 = new Order("adi", MenuItem.AMERICANO);
String o1Name = o1.getUser();
MenuItem o1MenuItem = o1.getMenuItem();
```

בדוגמא זו יצרנו הזמנה חדשה של המשתמשת עדי אשר הזמינה אמריקנו.

בנוסף לבנאי ושני ה getter-ים, הניחו כי Order מממשת את equals ואת hashCode בצורה תקינה.

כעת, נגדיר את המנשק IServicePolicy:

```
public interface IServicePolicy {

    /*
     * @post: getOrdersNum() = prev(getOrdersNum()) + 1
     * @post: checkIfOrderExist(order) == true
     */
    public void addOrder(Order order);

    public boolean checkIfOrderExist(Order order);

    /*
     * @post: prev(checkIfOrderExist(order)) == true implies
     *         getOrdersNum() = prev(getOrdersNum()) - 1
     * @post: prev(checkIfOrderExist(order)) == false implies
     *         getOrdersNum() = prev(getOrdersNum())
     * @post checkIfOrderExist(order) == false
     */
    public void removeOrder(Order order);

    public int getOrdersNum();

    /*
     * @pre: getOrdersNum() > 0
     * @post: getOrdersNum() = prev(getOrdersNum()) - 1
     */
    public void handleNextOrder();
}
```

המנשק חושף חמישה שירותים:

- getOrderNum תחזיר את מספר ההזמנות שנלקחו וטרם טופלו (מהרגע שהן נשלחות למטבח, הן לא מעניינות אותנו). השירות addOrder מוסיף את ההזמנה. שימו לב שאותו הלקוח יכול להזמין כמה מנות, ויכול להזמין גם את אותה המנה פעמיים.
- השירות checkIfOrderExist יחזיר true עבור הזמנה שקיימת ברשימת ההזמנות, ו false אחרת.
- השירות removeOrder יבטל הזמנה אם ההזמנה קיימת ולא טופלה. אם קיימות שתי הזמנות זהות, השירות יוריד אחת מהן (לא משנה באיזה סדר).
- השירות handleNextOrder יטפל במנה הבאה בתור. לצורך כל, עליו לקבוע מהי המנה הבאה לטיפול, לשלוח אותה למטבח, ולהוריד אותה מתור המנות הממתכות לטיפול. לאחר שהזמנה נשלחת לטיפול, היא נעלמת מהמערכת כך שלא ניתן להבדיל בינה לבין הזמנה שכלל לא נרשמה.
- השירות getOrdersNum יחזיר את מספר המנות שהוזמנו ועדיין לא טופלו.

סעיף א' (5 נקודות).

נגדיר את המחלקה האבסטרקטית AbstractPolicy באופן הבא:

```
public abstract class AbstractPolicy implements IServicePolicy{
    protected List<Order> ordersList = new ArrayList<>();

    public void addOrder(Order order) { ordersList.add(order); }

    public boolean checkIfOrderExist(Order order) {
        return ordersList.contains(order);
    }

    public void removeOrder(Order order) {
        ordersList.remove(order);
    }

    public int getOrdersNum() { return ordersList.size(); }

    public final void handleNextOrder() {

        Order nextOrder = getNextOrder();
        sendOrderToKitchen(nextOrder);
        removeOrder(nextOrder);

    }

    private void sendOrderToKitchen(Order order){
        //implementation provided
    }

    protected abstract Order getNextOrder();
```

השלימו המימוש של handleNextOrder. על מנת לבצע את שליחת ההזמנה, השתמשו בשירות sendOrderToKitchen שכבר מומש עבורכם. שימו לב ש getNextOrder האבסטרקטית מאפשרת למחלקה לתמוך במימושים שונים של מדיניות ניהול ההזמנות, כפי שניראה בסעיפים הבאים.

ממשו את המחלקה SimplePolicy שמספקת ניהול תור הזמנות בשיטת FIFO – ההזמנה הראשונה שנרשמה היא זו שתטופל ראשונה. במימוש זה ניתן להוסיף שדות אך אסור לדרוס שירותים שנורשו מ AbstractPolicy למעט getNextOrder.

```
public class SimplePolicy extends AbstractPolicy {
```

```
//add members here
```

```
protected abstract Order getNextOrder() {
```

```
return this.ordersList.get(0);
```

```
}
```

ממשו את המחלקה SemiRandomPolicy. מדיניות ניהול התור שלה: היא מטפלת לסירוגין במנה הראשונה בתור, ובמנה שנבחרת אקראית. כלומר, בפעם הראשונה מטופלת המנה שהוזמנה ראשונה, ובפעם הבאה מטופלת מנה אקראית מבין כל המנות שהוזמנו. במימוש זה ניתן להוסיף שדות אך אסור לדרוס שירותים שנורשו מ AbstractPolicy למעט getNextOrder.

לצורך הגרלת מספר אקראי העזרו במחלקה Random ע"פ דוגמת הקוד הבא:

```
Random r = new Random();
r.nextInt(10); //returns a random integer from 0 to 9.
```

```
public class SemiRandomPolicy extends AbstractPolicy {
```

```
//add members here
```

```
private boolean chooseRandom=false;
private Random rand = new Random()
```

```
protected abstract Order getNextOrder() {
    int nextIndex = chooseRandom ?
        rand.nextInt(this.ordersList.size()) :
        0;
    chooseRandom = ! chooseRandom;
    return this.ordersList.get(nextIndex);
```

```
}
```

סעיף ד' (13):

בסעיף זה נמש משמעות ניהול הזמנות אשר מתעדפת ביצוע של מנות מאותו הסוג אחת אחרי השניה.

המדיניות פועלת כך: נסתכל על המנה שהוזמנה הכי הרבה פעמים ונתחיל להכין את כל ההזמנות של מנה זו על פי סדר הזמנתן (אם יש שתי מנות שהוזמנו הכי הרבה פעמים, בחרו אחת מהן). נמשיך לטפל בכל ההזמנות של המנה הנבחרת, גם אם בינתיים נוספו הזמנות חדשות של מנות אחרות. אחרי שנסיים לטפל בכל ההזמנות של מנה הזו, נבחן שוב את רשימת ההזמנות המעודכנת, נבחר שוב את המנה הכי פופולרית מבין ההזמנות שקיימות, ונתחיל לטפל בהזמנות שלה.

דוגמא:

```
ByMenuItemPolicy p = new ByMenuItemPolicy();
p.addOrder(new Order("adi", MenuItem.AMERICANO));
p.addOrder(new Order("neta", MenuItem.SAGIT_SALD));
p.addOrder(new Order("shani", MenuItem.AMERICANO));
p.handleNextOrder(); // handling adi's order // line *
p.addOrder(new Order("raheli", MenuItem.EGG_AVO_BREAD));
p.addOrder(new Order("hadar", MenuItem.SAGIT_SALD));
p.addOrder(new Order("noa", MenuItem.AMERICANO));
p.addOrder(new Order("meytal", MenuItem.SAGIT_SALD));
p.handleNextOrder(); // handling shani's order // line **
p.handleNextOrder(); //handling noa's order // line ***
p.handleNextOrder(); //handling neta's order
p.handleNextOrder(); //handling hadar's order
p.handleNextOrder(); //handling meytals's order
p.handleNextOrder(); //handling raheli's order
```

בעת ביצוע שורה \* יש 2 הזמנות של אמריקנו ואחת של סלט שגית ולכן נקבע שההזמנות שיטופלו הן רק הזמנות של אמריקנו, על פי סדר ההזמנה. עדי תטופל ראשונה. בקריאה השניה ל handleNextOrder (שורה \*\*) נטפל בהזמנה של שני, ובקריאה השלישית (שורה \*\*\*) נטפל בהזמנה של נועה. שימו לב שלמרות שהתווספו בין הקריאות הזמנות נוספות, אנחנו ממשיכים לטפל בהזמנות של האמריקנו עד שיסתיימו גם אם בינתיים יש מנה אחת שהוזמנה יותר פעמים מהאמריקנו.

רק לאחר שסיימנו את כל ההזמנות של האמריקנו נוכל לעבור למנה אחרת. אחרי ביצוע שורה \*\*\* יש לנו 3 הזמנות של סלט שגית והזמנה אחת של egg avo bread. לכן, קודם יטופלו ההזמנות של סלט שגית (נטע, הדר ומיטל). לבסוף, נטפל בהזמנה של רחלי שהזמינה egg avo bread. בסיום הריצה הזו, כל ההזמנות במערכת טופלו.

השלימו את המימוש של המחלקה ByMenuItemPolicy. עליכם לספק מימוש עבור getNextOrder בנוסף, במידה שיש צורך לדרוס שירותים נוספים, עשו זאת. כמו כן, ניתן להוסיף שדות חדשים.

סיבוכיות נדרשת: כל השירותים שימושו כאן צריכים להיות ממומשים ב  $O(m)$  כאשר  $m$  הוא מספר המנות השונות שקיימות (כלומר, מספר האיברים ב MenuItem). זה אומר שמותר לכם לעשות זולאות על האיברים השונים ב MenuItem, אך עליכם להימנע מזולאות על כל ההזמנות במערכת.

```
public class ByMenuItemServicePolicy extends AbstractPolicy{

    //members here:

    Map<MenuItem, List<Order>> ordersGroupedByItem = new HashMap<>();
    MenuItem currDish = null;
```

```
protected abstract Order getNextOrder() {
```

```
    if (currDish == null ||
        ordersGroupedByItem.get(currDish).isEmpty()){
        for (MenuItem dish: ordersGroupedByItem.keySet()){
            if (currDish == null ||
                (ordersGroupedByItem.get(dish).size() >
                 ordersGroupedByItem.get(currDish).size())){
                currDish = dish;
            }
        }
    }
    return ordersGroupedByItem.get(currDish).get(0);
}
```

```
}
```

```
//overridden methods
```

```
public void addOrder(Order order){
    super.addOrder(order);
    List<Order> ordersForItem =
        ordersGroupedByItem.get(order.getMenuItem());
    if (ordersForItem == null){
        ordersForItem = new ArrayList<>();
        ordersGroupedByItem.put(order.getMenuItem(),
                                ordersForItem);
    }
    ordersForItem.add(order);
}

public boolean removeOrder(Order order){
    if (super.removeOrder(order)){
        ordersGroupedByItem.get(order.getMenuItem())
                            .remove(order);
        if (ordersGroupedByItem.get(order.getMenuItem())
                            .isEmpty()){
            ordersGroupedByItem.remove(order.getMenuItem());
        }
        return true;
    }
    return false;
}
```

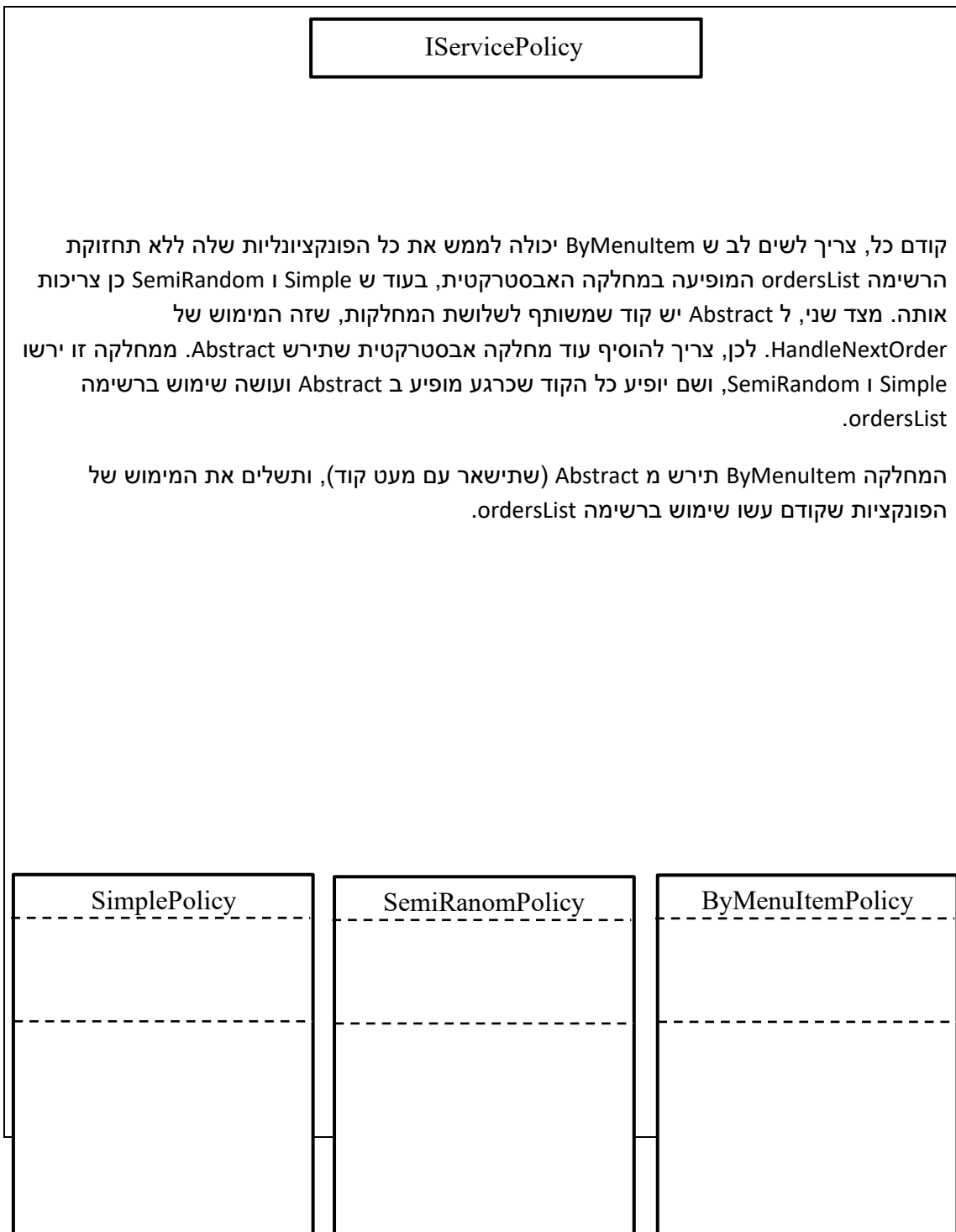
```
}
```

סעיף ה' (7 נק'):

הסטודנטית ברית טוענת שעיצוב המחלקות שהצגנו בסעיפים א-ד' הוא לא מספיק טוב, כיוון שחלק מהקוד ש AbstractPolicy מכילה לא באמת רלוונטי למחלקות שירותים ממנה ( Simple, ) (ByMenuItem | SemiRandom), מה שמייצר שמירת נתונים מיותרת ו\או ביצוע פעולות שלא לצורך.

בסעיף זה עליכם לתכנן מחדש את היררכיית המחלקות. שימו לב שאין צורך לספק מימושים! עליכם לפרט רק את המחלקות ואת קשרי הירושה שלהן (מי יורש ממי). לכל מחלקה עליכם לציין שלושה פרטים: שדות, פונקציות אבסטרקטיות ופונקציות שמומשו במחלקה. לנוחותכם, הוספנו מסגרות עבור שלושת המחלקות הקונקרטיות שצריכות להופיע בתחתית עץ הירושה (תוכלו להוסיף שם שדות ומתודות). בנוסף, בראש התרשים כבר מופיע IServicePolicy.

שימו לב – יתכנו מספר תשובות נכונות בשאלה זו כתלות במימוש שסיפקתם בסעיפים הקודמים.



## שאלה 2 (15 נק'):

סעיף א' (10 נק'):

בחלק זה עליכם לקרוא רשימה של משפטים מתוך קובץ. תוכן הקובץ יופיע בפורמט הבא:

```
<the
long
and
winding
road
<that
leads
to
your
door
```

כל מילה מופיעה בשורה נפרדת. ההפרדה למשפטים נעשית ע"י הסימן < הצמוד למילה הראשונה בכל משפט. הקובץ מכיל רק אותיות קטנות (lowercase) באנגלית, ירידות שורה ('\n') ואת הסימן <.

בדוגמא המצורפת, מופיעים סה"כ שני משפטים:

```
the long and winding road
that leads to your door
```

כל משפט ישמר כרשימה (List) של מילים. עבור דוגמא זו, נקבל שתי רשימות, ובכל אחת 5 מילים.

בעת טעינת הקובץ, עליכם לבצע בדיקת חוקיות על המשפטים. הקובץ יכול להיות ריק, אבל אסור לו להכיל משפטים ריקים. לדוגמא, קלט כזה הוא לא חוקי:

```
<the
long
<
<and
winding
```

המשפט השני אשר מופיע הוא משפט ריק. כאשר הקלט לא חוקי, עליכם לזרוק חריג מטיפוס FileNotFoundException עם מלל לבחירתכם. שגיאה זו תיזרק גם בכל מקרה של בעיה אחרת בקריאת הקובץ (למשל, הנתיב לקובץ לא תקין, וכו').

להלן המימוש של FileNotFoundException:

```
public class FileNotFoundException extends Exception {
    public FileNotFoundException(String message){ super(message); }
}
```

ניתן, אך לא חובה, להעזר ב BufferedReader באופן הבא:

```
BufferedReader br = new BufferedReader(new FileReader(fileName));
```

ממשו את הפונקציה loadSentences שמקבלת נתיב לקובץ ומחזירה רשימה של רשימות. כל תת רשימה תייצג משפט.



```

public static List<List<String>> loadSentences(String fileName)
    throws FileNotFoundException{

    BufferedReader br=null;
    try{
        br = new BufferedReader(new FileReader(fileName));
        String line;
        List<String> currSentence=null;
        List<List<String>> allSentences = new ArrayList<>();
        while ((line = br.readLine()) != null){
            line = line.trim();
            String word;
            if (line.startsWith("<")){
                currSentence = new ArrayList<>();
                allSentences.add(currSentence);
            }
            word = line.replace("<", "");
            if (line.length() > 0){
                currSentence.add(word);
            }
        }
        for (List<String> s : allSentences){
            if (s.size() == 0){
                throw new FileNotFoundException("Empty sentence!");
            }
        }
        return allSentences;
    }
    catch (Exception exp){
        throw new FileNotFoundException(e.getMessage());
    }
    finally{
        try { br.close(); }
        catch (IOException e) {
            throw new FileNotFoundException(e.getMessage());
        }
    }
}
}

```

סעיף ב' (5 נק')::

בסעיף זה עליכם לממש את הפונקציה `printFirstWordInSentence` אשר מקבלת שם של קובץ ומדפיסה את המילה הראשונה של כל משפט בשורה חדשה (עבור הקובץ בסעיף הקודם, יודפסו המילה the ואז המילה that).

השתמשו בפונקציה מסעיף א' בשביל קריאת הקובץ. אם טעינת הקובץ נכשלה מסיבה כלשהי, לא יודפס כלום. שימו לב שהפונקציה `printFirstWordInSentence` לא מצהירה על זריקת חריגים.

```

public static void printFirstWordInSentence(String fileName){
    try{
        List<List<String>> sentences = loadSentences(fileName);
        sentences.stream().map(x->x.get(0))
            .forEach(System.out::println);
    }
    catch(Exception exp){
    }
}
}

```

עמוד 10 מתוך 20

מספר זהות: \_\_\_\_\_ מספר מחברת: \_\_\_\_\_

מסגרת חירום:

**שאלה 3:**

```
public class Generic{
    public void func1(List<? super Number> l1,
                     List<? extends Number> l2){
        l1.add(l2.get(0));
    }

    public void func2(List<? extends Number> l1,
                     List<? super Number> l2){
        l1 = l2;
    }

    public void func3(List<? extends Number> l1,
                     ArrayList<? extends Number> l2){
        l1 = l2;
    }
}
```

לפניכם שלוש פונקציות הממומשות במחלקה Generic.

בחר\י בתשובה הטובה ביותר:

- א. רק פונקציה func1 מתקמפלת.
- ב. רק פונקציה func2 מתקמפלת.
- ג. רק פונקציה func3 מתקמפלת.
- ד. רק הפונקציות func1 + func2 מתקמפלות.
- ה. רק הפונקציות func1 + func3 מתקמפלות.**
- ו. רק הפונקציות func2 + func3 מתקמפלות.
- ז. כל הפונקציות מתקמפלות.
- ח. כל הפונקציות לא מתקמפלות.

נימוק:

**שאלה 4:**

איזו מבין הטענות נכונה? בחר\י בתשובה הטובה ביותר:

- א. על מנת להריץ כל תוכנית java מוכנה (למשל, תוכנית שמגיעה בקובץ jar) אנחנו חייבים להתקין JDK על המחשב.
- ב. אם נרצה לעשות שימוש בקובץ jar חיצוני בתוכנית שלנו, אנחנו חייבים להוסיף אותו לאותה התיקיה שבה מופיע הקוד שלנו, ואז ניתן לעשות בו שימוש מבלי לבצע שום שינוי בקוד ובהגדרות הפרוייקט.
- ג. רקורסיה אינסופית מנפחת תמיד את גודל המחסנית (Stack Overflow).**
- ד. מלבד תשובה זו כל התשובות לא נכונות.
- ה. מלבד תשובה זו יש יותר מתשובה נכונה אחת.

נימוק:

**שאלה 5:**

```
public class Base{
    public String myFunc(String str) throws IOException{
        return null;
    }
}

public class Sub extends Base{
    /*          // op1
    public String myFunc(String str) throws Exception{
        return null;
    } */

    /*          // op2
    public String myFunc(String str) throws EOFException{
        return null;
    } */

    /*          // op3
    public Object myFunc(Object obj){
        return null;
    } */
}
```

ב Sub יש שלושה מימושים ל myFunc שנמצאים בהערה. נרצה לבחון מה יקרה אם נוציא כל אחת מהפונקציות מההערה בנפרד. (תזכורת – EOFException יורשת מ IOException).

טענה 1: אם נוציא את op1 מהערה, הקוד של Sub יתקמפל.

טענה 2: אם נוציא את op2 מהערה, הקוד של Sub יתקמפל.

טענה 3: אם נוציא את op3 מהערה, הקוד של Sub יתקמפל.

בחר\י בתשובה הטובה ביותר:

- א. כל הטענות נכונות.
- ב. כל הטענות לא נכונות.
- ג. רק טענה 1 נכונה
- ד. רק טענה 2 נכונה
- ה. רק טענה 3 נכונה.
- ו. רק טענות 1+2 נכונות.
- ז. רק טענות 1+3 נכונות.
- ח. רק טענות 2+3 נכונות.

נימוק:

שאלה 6:

```

public class Q6 {
    public static void main(String[] args) {
        Stream<Integer> s= Stream.generate(new SmallNumbers());
        System.out.println(s.peek(x->System.out.print("x"))
            .filter(x->x%2== 0)
            .map(x->x/2)
            .peek(x->System.out.print("y"))
            .allMatch(x-> x < 3));}
}

public class SmallNumbers implements Supplier<Integer>{
    int currNum;
    public Integer get() {
        currNum++;
        if (currNum >= 6) {currNum = 0;}
        return currNum;
    }
}

```

מה יקרה בהרצת הקוד הבא?

בחר\י בתשובה הטובה ביותר:

- א. הקוד ידפיס את הרצף xy בלולאה אינסופית.
- ב. **הקוד ידפיס את הרצף xy בלולאה אינסופית.**
- ג. הקוד ידפיס את הרצף xy בלולאה אינסופית.
- ד. הקוד ידפיס xyxyxytrue ואז יעצור (הרצף xy מופיע 3 פעמים).
- ה. הקוד ידפיס xyxyxyxytrue ואז יעצור (הרצף xy מופיע 3 פעמים).
- ו. הקוד ידפיס xyxyxyxyxyxytrue ואז יעצור (הרצף xy מופיע 3 פעמים).
- ז. הקוד ירוץ בלולאה אינסופית ולא ידפיס כלום.
- ח. הקוד ידפיס false בסוף הריצה שלו.

נימוק:

**שאלה 7:**

```
public class Q7 {
    public static int[] modifyArray(int[] arr){
        arr[0] = 9;
        return arr;
    }

    public static void main(String[] args){
        int[] arr1 = {1,2,3};
        int[] arr2 = {1,2,3};
        System.out.print((arr1 == arr2) + " ");
        int[] arr3 = modifyArray(arr1);
        System.out.print((arr1[0] == arr2[0]) + " ");
        System.out.print(arr3 == arr1);
    }
}
```

מה יודפס בהרצת התוכנית?

- א. true true true
- ב. true true false
- ג. true false true
- ד. true false false
- ה. false true true
- ו. false true false
- ז. false false true
- ח. false false false

נימוק:

**שאלה 8:**

לפניכם מספר טענות בקשר ל enum-ים. בחר/י בתשובה הטובה ביותר:

- א. ניתן לרשת מ enum.
- ב. לא ניתן להשתמש ב enum בתוך בלוק switch/case.
- ג. ניתן להגדיר שדות ב enum.
- ד. מלבד תשובה זו כל התשובות לא נכונות.
- ה. מלבד תשובה זו יש יותר מתשובה נכונה אחת.

נימוק:

**שאלה 9:**

```
public class Gen<T>{
    public static T outerT; /*

    public T getAnotherT() { return new T();} //#

    public class Inner{
        public T innerT; //$
    }
}
```

אילו מבין השורות המסומנות ב \*, # ו \$ מתקמפלות? בחר\י בתשובה הטובה ביותר:

- א. רק השורה \* מתקמפלת.
- ב. רק השורה # מתקמפלת.
- ג. רק השורה \$ מתקמפלת.
- ד. רק השורות \* ו # מתקמפלות.
- ה. רק השורות \* ו \$ מתקמפלות.
- ו. רק השורות # ו \$ מתקמפלות.
- ז. כל השורות מתקמפלות.
- ח. כל השורות לא מתקמפלות.

נימוק:

**שאלה 10:**

```
public static void func(B b){
    A c = (A)b //originally was (B)b, changed after the test
}
```

לפניכם שלוש טענות על הקוד המצורף:

- טענה 1: הקוד יתקמפל לכל A שהיא מחלקה קונקרטית (לא אבסטרקטית) ולכל B שהוא ממשק.
- טענה 2: הקוד יתקמפל וירוץ ללא שגיאות זמן ריצה על Casting לכל A שירשת מ B.
- טענה 3: הקוד יתקמפל לכל A ו B שירשות ממחלקה C.

בחר\י בתשובה הטובה ביותר:

- א. כל הטענות נכונות.
- ב. כל הטענות לא נכונות.
- ג. רק טענה 1 נכונה
- ד. רק טענה 2 נכונה
- ה. רק טענה 3 נכונה.
- ו. רק טענות 1+2 נכונות.
- ז. רק טענות 1+3 נכונות.
- ח. רק טענות 2+3 נכונות.

נימוק:

**שאלה 11:**

```

/* class Box<T extends Comparable<T>> implements Comparable<T> */ //op1
/* class Box<T> implements Comparable<Box<T>> */ //op2
/* class Box<T extends Comparable> implements Comparable */ //op3

public class Box<T> ***** {
    T t;

    public Box(T t){
        this.t = t;
    }

    public void func(Box<T> otherBox){
        System.out.println(this.t.compareTo(otherBox.t)); /**
    }

    public static void main(String[] args){
        Box<String> b1 = new Box("abc");
        Box<String> b2 = new Box("dce");
        System.out.println(b1.compareTo(b2));
    }

    /* other code */
}

```

לפניכם הקוד של המחלקה הגנרית Box. ההצהרה על Box מכילה חלק מוסתר בכוכביות, ואותו תצטרכו להשלים. במחלקה עצמה מופיע מימוש של שתי פונקציות: main ו func. נרצה להשלים את ההצהרה על Box כך שהקוד של main ו func יתקמפל.

שלושת האופציות להשלמת ההצהרה על Box מופיעות בהערה מעל מימוש המחלקה. הניחו כי עבור כל אחת מהאופציות המוצעות מימוש הפונקציה של המנשק אותו Box מממשת מופיע במקום המסומן ב /\* other code\*/, והוא תקין. במילים אחרות, עבור כל שינוי, עליכם לבדוק אם func ו main מתקמפלות, ולהניח שכל ששאר הקוד תקין.

בחרו בתשובה הטובה ביותר:

- א. רק op1 מתאימה.
- ב. רק op2 מתאימה.
- ג. רק op3 מתאימה.
- ד. רק op1+op2 מתאימות.
- ה. רק op1+op3 מתאימות.
- ו. רק op2+op3 מתאימות.
- ז. כל האופציות מתאימות.
- ח. כל האופציות לא מתאימות.



נימוק:

**שאלה 12:**

```
public final class Q12 {
    public final int i = func();

    public final Q12 f(){ return new Q12(); }    /**

    public int func() {    /**#
        i = 1;
        return i;
    }

    public void foo() {    /**$
        Q12 q = new Q12(){
            public void func(){
                System.out.println("override");
            }
        };
    }
}
```

לפניכם מחלקה עם שלוש פונקציות. עליכם לבחון כל פונקציה בנפרד ולהחליט אם היא מתקמפלת. בחר\י בתשובה הטובה ביותר:

- א. רק הפונקציה \* מתקמפלת.
- ב. רק הפונקציה # מתקמפלת.
- ג. רק הפונקציה \$ מתקמפלת.
- ד. רק הפונקציות \* ו # מתקמפלות.
- ה. רק הפונקציות \* ו \$ מתקמפלות.
- ו. רק הפונקציות # ו \$ מתקמפלות.
- ז. כל הפונקציות מתקמפלות.
- ח. כל הפונקציות לא מתקמפלות.

נימוק:

שאלה 13:

```

public interface MyI<T> {
    public T func(T x);
}

public class MyIntI implements MyI<Integer>{
    public Integer func(Integer x) { return x*2;}
}

public class Test{

    public static void runTest(MyI<Integer> myI) {
        System.out.print(myI.func(3));
    }

    public static void main(String[] args) {
        // runTest((x,y)->x+y); //op1

        // runTest(x->new MyIntI()); //op2

        // runTest(x->x*2); //op3
    }
}

```

לפניכם שלוש אופציות להשלים את מימוש הפונקציה main, כאשר כל אופציה נבחנת בנפרד. אילו מבין המימושים מתקמפלי? בחר\י בתשובה הטובה ביותר.

- א. רק op1 מתקמפלת.
- ב. רק op2 מתקמפלת.
- ג. רק op3 מתקמפלת.
- ד. רק op1 ו op2 מתקמפלות.
- ה. רק op1 ו op3 מתקמפלות.
- ו. רק op2 ו op3 מתקמפלות.
- ז. שלושת האופציות מתקמפלות.
- ח. שלושת האופציות לא מתקמפלות.

נימוק:

---

**public interface Map<K,V>**

void	clear() Removes all of the mappings from this map (optional operation).
boolean	containsKey(Object key) Returns true if this map contains a mapping for the specified key.
Set<Map.Entry<K,V>>	entrySet() Returns a Set view of the mappings contained in this map.
V	get(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
V	getOrDefault(Object key, V defaultValue) Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
boolean	isEmpty() Returns true if this map contains no key-value mappings.
Set<K>	keySet() Returns a Set view of the keys contained in this map.
V	put(K key, V value) Associates the specified value with the specified key in this map. Returns the previous value associated with key, or null if there was no mapping for key.
V	remove(Object key) Removes the mapping for a key from this map if it is present (optional operation). Returns the previous value associated with key, or null if there was no mapping for key.
int	size() Returns the number of key-value mappings in this map.
Collection<V>	values() Returns a Collection view of the values contained in this map.

**public interface Set<E> extends Collection<E>**

boolean	add(E e) Adds the specified element to this set if it is not already present.
void	clear() Removes all of the elements from this set (optional operation).
boolean	contains(Object o) Returns true if this set contains the specified element.
boolean	isEmpty() Returns true if this set contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this set.
boolean	remove(Object o) Removes the specified element from this set if it is present.
int	size() Returns the number of elements in this set (its cardinality).

**public interface List<E> extends Collection<E>**

boolean	add(E e) Appends the specified element to the end of this list. Always returns true.
---------	---

void	<code>clear()</code> Removes all of the elements from this list.
boolean	<code>contains(Object o)</code> Returns true if this list contains the specified element.
E	<code>get(int index)</code> Returns the element at the specified position in this list.
int	<code>indexOf(Object o)</code> Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
Iterator<E>	<code>iterator()</code> Returns an iterator over the elements in this list in proper sequence.
boolean	<code>remove(Object o)</code> Removes the first occurrence of the specified element from this list, if it is present. Returns true if this list contained the specified element
int	<code>size()</code> Returns the number of elements in this list.
default void	<code>sort(Comparator&lt;? super E&gt; c)</code> Sorts this list according to the order induced by the specified Comparator.

**public final class String**

char	<code>charAt(int index)</code> Returns the char value at the specified index.
int	<code>length()</code> Returns the length of this string.
String[]	<code>split(String regex)</code> Splits this string around matches of the given regular expression.
boolean	<code>startsWith(String prefix)</code> Tests if this string starts with the specified prefix.

**public class BufferedReader**

	<code>BufferedReader(Reader in)</code> Creates a buffering character-input stream that uses a default-sized input buffer
void	<code>close()</code> Closes the stream and releases any system resources associated with it.
int	<code>read()</code> Reads a single character. Returns the character read, as an integer, or -1 if the end of the stream has been reached. Throws <code>IOException</code> if an I/O error occurs.
String	<code>readLine()</code> Reads a line of text. Returns A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached. Throws <code>IOException</code> if an I/O error occurs.

**public class FileReader (extends Reader)**

	<code>FileReader(File file)</code> Creates a new <code>FileReader</code> , given the <code>File</code> to read from. Throws <code>FileNotFoundException</code> (extends <code>IOException</code> ) if the file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading.
--	---