בחינה בתוכנה 1

סמסטר ב' תשפ"א, מועד א', 13 ביולי 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 לכתיבה נכונה ב Java.

<u>מבנה הבחינה:</u>
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 מכנית ובין אלקטרונית או בכל דרך אחרת כל חלק שהוא מטופס הבחינה.

בהצלחה!

חלק א':

שאלה 1 (35 נקודות):

בשאלה זו נרצה לממש מפה (Map) גנרית המאפשרת גישות דו כיווניות, גם לפי המפתח (key) וגם לפי הערך (value). המימוש יעשה על פי המנשק הבא:

```
public interface IBiDirMap<K,V> extends Iterable<K>{

    /* @pre: key != null, value != null */
    public void put(K key, V value);

    /* @pre: key != null
     * @post: $ret is the previous value associated with key,
     *         or null if there is no mapping for key */
    public V removeKey(K key);

    /* @pre: key != null
     * @post: $ret is the value to which the specified key is mapped,
     *         or null if there is no mapping for the key */
    public V getValueForKey(K key);

    /* @pre: value != null
     * @post: $ret is the set of all keys that are mapped to value
     * @post: $ret != null */
    public Set<K> getKeysForValue(V value);

    /* @post: $ret the set of all keys in the map, $ret != null */
    public Set<K> keySet();

    /* @post: $ret the set of all values in the map, $ret != null */
    public Set<V> valueSet();

}
```

נשים לב שהמנשק מבטא את חוסר הסימטריה בין מפתחות לערכים. מפתח ממופה תמיד לערך יחיד, לכן עבור כל מפתח (key) הפונקציה getValueForKey תחזיר ערך אחד. לעומת זאת, ערך בודד יכול להתאים ליותר ממפתח אחד, ולכן getKeysForValue מחזירה קבוצה של מפתחות. במידה שהערך לא קיים במילון, תחזור קבוצה ריקה.

להלן דוגמא לאופן השימוש המבוקש במנשק:

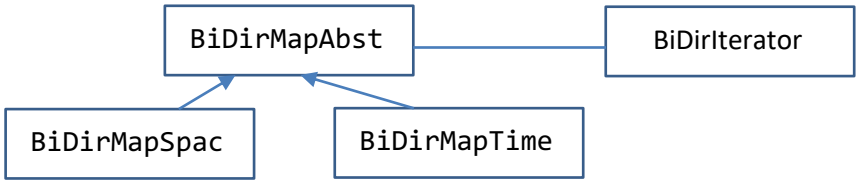
```
IBiDirMap<Integer,String> map = new BiDirMapSpace<>();
map.put(1, "a");
map.put(2, "b");
map.put(3, "a");
System.out.println(map.getValueForKey(1)); // "a"
System.out.println(map.getKeysForValue("a")); //[1, 3]
System.out.println(map.keySet()); //[1, 2, 3]
System.out.println(map.valueSet()); //[a, b]
map.put(1, "c");  //*
System.out.println(map.getKeysForValue("a")); //[3]
map.removeKey(3); //**
System.out.println(map.getKeysForValue("a")); //[]
```

הערות לדוגמת הריצה:

- שורה *: ההוספה השניה של המפתח 1 (עם הערך c) דורסת את הערך "a" שמופה קודם ל 1, לכן, כשנדפיס שוב את המפתחות שמתאימים ל "a", 1 כבר לא יודפס.
- שורה **: אחרי שמחקנו את המפתח 3, שום מפתח ללא ממופה יותר ל "a" ולכן קבוצת המפתחות שלו היא קבוצה ריקה.
- מכיוון ש Set אינו מסודר, סדר ההדפסה שמופיע בדוגמא זו אינו מחייב.

בשאלה זו יש שלושה סעיפים: בשני הסעיפים הראשונים תממשו את IBiDirMap בשני אופנים – הראשון, חסכוני בזכרון ובזבזני בזמן ריצה (סעיף א'). המימוש השני הוא בזבזני בזכרון אך חסכוני בזמן ריצה (סעיף ב'). בסעיף השלישי תממשו את האיטרטור BiDirIterator.

מבנה המחלקות ניראה כך:



המחלקה האבסטרקטית BiDirMapAbst שמכילה קוד משותף לשתי המחלקות שתממשו כבר מומשה עבורכם בקוד המצורף. קוד זה מטפל בשימוש רגיל ב Map – גישה ממפתח לערך.

```
public abstract class BiDirMapAbst<K,V> implements IBiDirMap<K,V>{

    private Map<K,V> keyToValueMap = new HashMap<>();

    public void put(K key, V value){   keyToValueMap.put(key, value);}

    public V removeKey(K key){ return keyToValueMap.remove(key); }

    public V getValueForKey(K key){ return keyToValueMap.get(key);}

    public Set<K> keySet(){ return keyToValueMap.keySet(); }

    public Iterator<K> iterator(){
        return new BiDirIterator<K,V>(this); //last section
    }

    public abstract Set<K> getKeysForValue(V value);

    public abstract Set<V> valueSet();


}
```

הפונקציות getKeysForValue ו valueSet הוגדרו במנשק ולא מומשו במחלקה האבסטרקטית, כך שהן מופיעות כאן לצורך קריאות בלבד (כזכור, לא חובה להוסיף חתימות אלה למחלקה האבסטרקטית). כל מחלקה לא אבסטרקטית שתירש מ BiDirMapAbst תצטרך לממש את הפונקציות האלה.

שימו לב:

1. כל הסעיפים הם עצמאיים ובלתי תלויים אחד בשני.
2. כל אחת מהמחלקות ממומשת בקובץ נפרד, כך שאין להן גישה לשדה keyToValueMap של BiDirMapAbst. את כל הפונקציונליות עליכם לממש באמצעות קריאות לפונקציות של המנשק.

סעיף א (8 נק'):

ממשו את המחלקה BiDirMapSpace אשר מממשת מפה דו כיוונית חסכונית בזכרון. מחלקה זו לא שומרת שום מבנה נתונים נוסף מלבד השדה keyToValueMap שנורש מ BiDirMapAbst, ותהליך בניית המיפוי ההפוך (ערך לקבוצת מפתחות) נעשה בזמן ריצה בפונקציות הרלוונטיות.

עליכם לממש את הפונקציה getKeysForValue שלא מומשה ב BiDirMapAbst. בשאלה זו אין צורך לממש את valueSet. בנוסף, עליכם לדרוס פונקציות שכן מומשו אך נדרש להן שינוי (במידה שיש כאלה). ניתן, אך לא חובה, להוסיף מימוש של בנאי ללא פרמטרים, אם יש בזה צורך.

המימוש צריך לעשות שימוש חוזר בקוד, במידת האפשר. כמו כן, אתם רשאים להוסיף שירותי עזר במידת הצורך.

```
public class BiDirMapSpace<K,V> extends BiDirMapAbst<K,V> {

    public Set<V> valueSet() { /* no need to implement */}

    public Set<K> getKeysForValue(V value){
```

```
        Set<K> allKeys = new HashSet<>();
        for (K key: this.keySet()){
            if (value.equals(getValueForKey(key))){
                allKeys.add(key);
            }
        }
        return allKeys;
```

```
    }

// overridden methods here
```

```

```

```
}
```

<u>סעיף ב (15 נק'):</u>

ממשו את המחלקה BiDirMapTime אשר מממשת מפה דו כיוונית חסכונית בזמן ריצה. מחלקה זו יכולה להחזיק מבני נתונים נוספים, לבחירתה, על מנת שבקריאה לכל אחת מהמפעולות שהוגדרו במנשק לא יהיה מעבר בלולאה על כל האיברים במילון (במילים אחרות, סיבוכיות קטנה מ O(n).

עליכם לממש את הפונקציות getKeysForValue ו valueSet שלא מומשו ב BiDirMapAbst. בנוסף, עליכם לדרוס פונקציות שכן מומשו אך נדרש להן שינוי (במידה שיש כאלה). ניתן, אך לא חובה, להוסיף מימוש של בנאי ללא פרמטרים, אם יש בזה צורך.

גם כאן המימוש צריך לעשות שימוש חוזר בקוד, במידת האפשר. כמו כן, אתם רשאים להוסיף שירותי עזר במידת הצורך.

```
public class BiDirMapTime<K,V> extends BiDirMapAbst<K,V>{

    //add members here:
    Map<V, Set<K>> valuesToKeys = new HashMap<>();


    public Set<V> valueSet(){

        return valuesToKeys.keySet();


    }

    public Set<K> getKeysForValue(V value){

        return valuesToKeys.getOrDefault(value, new HashSet<>());



    }

    // overridden methods here

    public void put(K key, V value){

        // remove previous value
        removeKey(key);

        // put new key (possible also a new value)
        Set<K> keysForVal = getKeysForValue(value);
        keysForVal.add(key);
        valuesToKeys.put(value, keysForVal);
        super.put(key, value);
    }
```

```
  public V removeKey(K key){
     V oldVal = super.removeKey(key);
     if (oldVal != null){
        // remove the key from the keys set for value
        valuesToKeys.get(oldVal).remove(key);
           if (valuesToKeys.get(oldVal).isEmpty()){
              /* remove the value entirely if key was
                 the only key mapped for this value */
              valuesToKeys.remove(oldVal);
           }
     }
     return oldVal;
  }
```

}

<u>סעיף ג (12 נק)':</u>

כעת עליכם לממש את המחלקה BiDirIterator שבה עושה שימוש המחלקה BiDirMapAbst. האיטרטור יחזיר את המפתחות של BiDirMapAbst בסדר הבא: הסדר הפנימי בין המפתחות לא משנה, אבל הם צריכים להיות מקובצים על פי הערכים שלהם. כלומר: עבור מפתח key שממופה ל value כלשהו, אם האיטרטור החזיר את המפתח key, באיטרציה הבאה יחזור מפתח אחר שממופה ל value. אם אין כזה (כלומר, כל המפתחות המתאימים לאותו ה value כבר הוחזרו), עוברים למפתח הבא.

לדוגמא:

```
IBiDirMap<Integer,String> map = new BiDirMapSpace<>();
map.put(1, "a");
map.put(2, "b");
map.put(3, "a");
map.put(4, "b");
map.put(5, "a");
map.put(6, "c");
for(Integer i : map){
   System.out.print(i + " ");
}
```

בדוגמא זו נוצרת מפה עם 3 ערכים שונים. עבור "a" יש שלושה מפתחות שונים: 1,3,5. עבור "b" קיימים המפתחות 2,4 ול "c" ממופה מפתח יחיד - 6. לכן, סדר ההדפסה של האיטרטור יתחשב בכך ש 1,3,5 צריכים להופיע אחד אחרי השני (בסדר כלשהו), וכך גם 2 ו4. מעבר לזה, אין חשיבות לסדר ההדפסה.

פלט אפשרי של האיטרטור יכול להיות 6 4 2 5 3 1. גם הסידור 2 4 6 1 3 5 מתאים, כיוון ש [5,3,1] ו [4,2] מקובצים יחד. לעומת זאת הסידור 6 5 4 3 2 1 לא מתאים.

בשלד הפתרון המצורף נוספו עבורכם שדות שבהן כדאי שבהם כדאי להשתמש. אתם רשאים להוסיף שדות נוספים ולא לעשות שימוש בשדות שהגדרנו.

```
public class BiDirIterator<K, V> implements Iterator<K> {

    BiDirMapAbst<K, V> biDirMap;
    Iterator<V> valuesIt;
    Iterator<K> keysIt;
```

```

```

```
    public BiDirIterator(BiDirMapAbst<K, V> biDirMap) {
```

```
       this.biDirMap = biDirMap;
       valuesIt = biDirMap.valueSet().iterator();
       keysIt = valuesIt.hasNext() ? getKeysIterForNextValue()
                                   : new HashSet<K>().iterator();
```

```
    }
    public boolean hasNext() {
```

```
 return valuesIt.hasNext() || keysIt.hasNext();
```

```

```

```
    }

    public K next() {
```

```
      if (!keysIt.hasNext()){
         keysIt = getKeysIterForNextValue()
      }
      return keysIt.next();
   }



   private Iterator<K> getKeysIterForNextValue(){
      V nextValue = valuesIt.next();
      return biDirMap.getKeysForValue(nextValue).iterator();
   }
```

```

```

```
    }
}
```

שאלה 2 (15 נק׳):

בשאלה זו נכתוב קוד אשר מחפש דמיון בין ספרים ע״י השוואת המילים השונות שמופיעות בכל ספר.

התוכן של כל ספר מיוצג ע״י רשימת משפטים. כל משפט מכיל מילים ב lowercase בלבד, וכל שתי מילים מופרדות ע״י רווח בודד, כך שאין צורך לעשות שום נקיון על הטקסט מלבד ההפרדה למילים.

עליכם לממש 3 פונקציות:

1. הפונקציה extractWords אשר מקבלת רשימת משפטים ומחזירה קבוצה (Set) עם כל המילים השונות (ללא חזרות) מכל המשפטים. השירות יעשה מעבר יחיד על כל משפט.
2. הפונקציה similarityScore מקבלת שתי קבוצות (Set) של מילים כאשר כל קבוצה מכילה את המילים השונות של ספר כלשהו. הפונקציה מחזירה את מספר המילים השונות המשותפות לשני הספרים. מספר זה מייצג את מדד הדמיון בין שני הספרים.
3. הפונקציה analyzeSimilarityForBook מקבלת שני פרמטרים: books - מפה (Map) של שמות ספרים לרשימת המשפטים שלהם, ו bookName - שם של ספר שמופיע כמפתח ב books. הפונק׳ תחזיר את רשימת כל שמות הספרים מ books ממויינים בסדר יורד של דמיון ל bookName, לא כולל bookName, על פי מדד הדמיון שהוגדר בסעיף 2. אם יש שני ספרים עם מדד דמיון זהה ל bookName, אין חשיבות לסדר ביניהם ברשימה המוחזרת.

להלן דוגמת ריצה עבור שלושה ספרים:

```
Map<String, List<String>> books = new HashMap<>();
books.put("book1", Arrays.asList("i love icecream", "i love cakes"));
books.put("book2", Arrays.asList("love love me do"));
books.put("book3", Arrays.asList("do you like cheesecake", "me too"));
Set<String> set1 = new HashSet<>();
Set<String> set2 = new HashSet<>();
set1.addAll(Arrays.asList("a", "b", "c"));
set2.addAll(Arrays.asList("a", "b", "d", "e"));

System.out.println(extractWords(books.get("book1")));
// [i,love, icecream, cakes]
System.out.println(similarityScore(set1, set2));
// 2
System.out.println(analyzeSimilarityForBook(books, "book2"));
// [book3, book1]
```

הפלט עבור כל הדפסה מופיע מתחתיה. בהדפסה הראשונה אין חשיבות לסדר המילים בקבוצה. בהדפסה השלישית, מדד הדמיון של book3 עם book2 הוא 2 (שתי מילים משותפות: do, me). מדד הדמיון של book1 עם book2 הוא 1 (מילה אחת משותפת: love). לכן, book3 יופיע לפני book2.

```
/*   @pre: sentences.size() > 0;     */
public static Set<String> extractWords(List<String> sentences){

    Set<String> res = new HashSet<>();
    for (String sent: sentences){
       for(String word : sent.split(" ")){
          res.add(word);
       }
    }
    return res;



}
```

```
    /*     @pre: book1Words.size() > 0, book2Words.size() > 0    */
    public static int similarityScore(Set<String> book1Words,
                                      Set<String> book2Words){
```

```
        return book1Words.stream().
                    filter(x->book2Words.contains(x))
                    .collect(Collectors.toSet()).size();
```

```
    }

    /*  @pre: books.size() > 2, books.containsKey(bookName) */
    public static List<String> analyzeSimilarityForBook(
                Map<String, List<String>> books, String bookName){
```

```
// all words for book name
Set<String> wordsForBookName = extractWords(books.get(bookName));
Map<String, Integer> bookToSimScore = new HashMap<>();

// populate book to score map
for (String currBookName: books.keySet()){
   if (!currBookName.equals(bookName)){
         Set<String> currS =
                         extractWords(books.get(currBookName));
         int simScore = similarityScore(wordsForBookName, currS);
         bookToSimScore.put(currBookName, simScore);
   }
}

List<String> res = new ArrayList<>();
res.addAll(bookToSimScore.keySet());
// sort names list by score in a descending order
res.sort((x,y)-> -Integer.compare(bookToSimScore.get(x),
                                  bookToSimScore.get(y)));
return res;
```

```
    }
```

מסגרת חירום:

חלק ב׳ (55 נקודות):

שאלה 3:

```
public static void main(String[] args) throws Exception {
    int x = 0;
    int[] a = { 1, 2, 3, 4 };
    for (int i : a) {
        try {
            if (i % 2 == 0) { throw new Exception();}
            if (i % 3 == 0) { throw new NullPointerException(); }
            x++;
        }
        catch (RuntimeException e) { x = x + 2;}

        catch (Exception e) { x++; }

        finally { x++; }
    }
    System.out.println(x);
}
```

מה יקרה בהרצת התוכנית הבאה? בחר\י בתשובה הטובה ביותר:

א. התוכנית תעוף בגלל שגיאה שלא טופלה, ולא יודפס שום פלט.
ב. ריצת התוכנית תסתיים בהצלחה ובסיומה יודפס 6
ג. ריצת התוכנית תסתיים בהצלחה ובסיומה יודפס 7
ד. ריצת התוכנית תסתיים בהצלחה ובסיומה יודפס 8
ה. ריצת התוכנית תסתיים בהצלחה ובסיומה יודפס 9
ו. ריצת התוכנית תסתיים בהצלחה ובסיומה יודפס 11

נימוק:

שאלה 4:

```
public class Generic<T>{
    public void func1(Set<T> s1, Set<T> s2) {};
    public void func2(Set<?> s1, Set<?> s2) {};
    public void func3(Set<Object> s1, Set s2) {};
}
```

לפניכם שלוש פונקציות הממומשות במחלקה Generic. נרצה לדעת האם ניתן להחליף כל שימוש בפונק׳ אחת ע״י קריאה לפונקציה אחרת מבלי לבצע כל שינוי בקוד מלבד החלפת הפונקציה.

טענה 1: ניתן להחליף כל קריאה ל func2 בקריאה ל func1, והקוד ימשיך להתקמפל.

טענה 2: ניתן להחליף כל קריאה ל func1 בקריאה ל func2 והקוד ימשיך להתקמפל.

טענה 3: ניתן להחליף כל קריאה ל func2 בקריאה ל func3 והקוד ימשיך להתקמפל.

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 5:

```
public class A{
    public A(String str){  }
}

public class B extends A{
    public B(int i){
        //super(); // op1
        //this(); //op2
        //this("abc"); //op3
    }

    public B(String str){ super(str); }
}
```

לפניכם קוד המחלקות A ו B. בבנאי של מחלקה B יש שלוש אופציות להשלמת המימוש. נרצה לבחון איך המחלקה B מתנהגת במימוש הנתון, ואיך היא תתנהג אם נוציא מהערה כל אחת מבין השורות בנפרד (כלומר, בכל פעם נוציא מההערה שורת קוד אחת מבין אופציות 1-3).

בחר\י בתשובה הטובה ביותר:

א. הקוד מתקמפל כמו שהוא. אם נוציא את אופציה 1 מההערה, הקוד ימשיך להתקמפל.
ב. הקוד מתקמפל כמו שהוא. אם נוציא את אופציה 2 מההערה, הקוד ימשיך להתקמפל.
ג. הקוד מתקמפל כמו שהוא. אם נוציא את אופציה 3 מההערה, הקוד ימשיך להתקמפל.
ד. הקוד לא מתקמפל. אם נוציא את אופציה 1 מההערה, הקוד יתקמפל.
ה. הקוד לא מתקמפל. אם נוציא את אופציה 2 מההערה, הקוד יתקמפל.
ו. הקוד לא מתקמפל. אם נוציא את אופציה 3 מההערה, הקוד יתקמפל.
ז. אם נוציא כל אחת מהאופציות 1-3 מהערה, הקוד לא יתקמפל.
ח. קיימת יותר מאופציה אחת כך שאם נוציא אותה מההערה הקוד יתקמפל.

נימוק:

שאלה 6:

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

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

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

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

ב Sub יש שלושה מימושים ל myFunc שנמצאים בהערה. נרצה לבחון מה יקרה אם נוציא כל אחת מהפונקציות מההערה בנפרד.

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:

<u>שאלה 7:</u>

לפניכם הצעה למימוש 3 מערכות תוכנה ע״י שימוש בתבנית העיצוב Bridge.

<u>מערכת 1</u> - מערכת המנהלת תזמורת. התזמורת שלנו מורכבת משני סוגים של כלי נגינה: כלי נשיפה וכלי מיתר, כאשר כל כלי נגינה משתייך רק לאחת מהקטגוריות. בנוסף לכלי נגינה נממש את המחלקה שמייצגת את התזמורת עצמה. נשתמש ב Bridge בשביל לאפשר לתזמורת לעבוד בצורה שקופה עם כל כלי הנגינה, מבלי לדעת אם כלי כלשהו הוא כלי מיתר או כלי נשיפה.

<u>מערכת 2</u> - מערכת עזר לפרשנים של משחקי היורו (Euro) בכדורגל. המערכת שלנו תכיל מבנה נתונים ענק של כל מני נתונים סטטיסטיים מאירי עיניים כמו ״מתי בפעם האחרונה ניצחה בלגיה את איטליה במשחק יורו״. נרצה לוודא שהנתונים נטענים פעם אחת בלבד בתחילת התוכנית, ולא בכל פעם שמישהו ירצה לגשת אליהם. לצורך כך, נשתמש בתבנית Bridge על מנת שכל הקוד שניגש לנתונים יעבור דרך מקום אחד בלבד.

<u>מערכת 3</u> - מערכת לניהול גן חיות. בגן החיות קיימים בעלי חיים מהמינים הבאים: יונקים, עופות וזוחלים ולכל מין יש מאפיינים יחודיים. בנוסף לחלוקה למינים, ישנה חלוקה של בעלי החיים לפי ההתנהגות במהלך היום. ישנן חיות שפעילות בעיקר בלילה (ינשופים, תיקנים), וישנן חיות שפעילות בעיקר ביום (נשרים, פילים). נשתמש בתבנית העיצוב Bridge בשביל לבטא עבור כל חיה גם מאפיינים הקשורים למין שלה, וגם את ההתנהגות שלה בהקשר של פעילות יום\לילה.

באיזו משלושת המערכות נעשה שימוש נכון ב Bridge?

א. רק במערכת 1.
ב. רק במערכת 2.
ג. רק במערכת 3.
ד. רק במערכות 2+1.
ה. רק במערכות 3+1.
ו. רק במערכות 3+2.

נימוק:

<u>שאלה 8:</u>

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

א. המפרש (Interpreter) קורא קבצי Java ומייצר קבצי byte code .
ב. ב Java לא מובטחת תאימות לאחור – תוכנית שנכתבה ב Java5 לא בהכרח תרוץ ב Java8.
ג. על מנת להריץ תוכנית Java על מחשב כלשהו לא נדרשת התקנה מיוחדת. התקנת Java נדרשת רק על בשלב הפיתוח.
ד. מלבד תשובה זו כל התשובות לא נכונות.
ה. מלבד תשובה זו יש יותר מתשובה נכונה אחת.

נימוק:

שאלה 9:

```
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->System.out.println(x*2)); //op3
    }
}
```

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

א. רק op1 מתקמפלת.
ב. רק op2 מתקמפלת.
ג. רק op3 מתקפמלת.
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:

שאלה 10:

מה יודפס בהרצת התוכנית Sub?

```
public class Base{
    public int func(){
        return 2 + foo();
    }

    private int foo(){
        return 5 + moo();
    }

    public int moo(){
        return 1;
    }
}

public class Sub extends Base{
    public int goo(){
        return func() + foo();
    }

    public int foo(){
        return 3;
    }

    public int moo(){
        return 2;
    }

    public static void main(String[] args){
        Sub sub = new Sub();
        System.out.println(sub.goo());
    }
}
```

בחר\י בתשובה הטובה ביותר:

א. 7
ב. 8
ג. 9
ד. 10
ה. 11
ו. 12
ז. 13

נימוק:

שאלה 11:

```
public class Q11 {
    public static int[] modifyArray(int[][] arrOfArr){
        arrOfArr[1] = new int[] {1,2,3};
        return arrOfArr[1];
    }

    public static void main(String[] args){
        int[][] arrOfArr = new int[3][];
        arrOfArr[0] = new int[] {1,2,3};
        arrOfArr[1] = new int[] {5,6,7};
        int[] arr1 = modifyArray(arrOfArr);
        System.out.print((arrOfArr[2] == null) + " ");
        System.out.print((arrOfArr[0][0] == arrOfArr[1][0]) + " ");
        System.out.print(arrOfArr[0] == 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 12:

נתונה המחלקה X שבתוכה מוגדר השדה הסטטי s:

```
public class X{
    public static String s = "abc";

    public void func(){ /* some code here */ }

    public static void foo(){ /* some code here */ }

    /* The rest of the code */
}
```

איזו טענה מבין הטענות הבאות לא נכונה?:

א. גם אם יצרנו יותר ממופע אחד של X, השדה הסטטי s נוצר פעם אחת בלבד.
ב. הקוד המופיע ב func יכול לעשות שימוש ב s.
ג. הקוד המופיע ב foo יכול לעשות שימוש ב s.
ד. s נשמר על ה heap.
ה. אם s לא מאותחל בשורה שבה הוא מוגדר, הוא יקבל את הערך null.
ו. שדה מטיפוס static לא יכול להיות מוגדר כ private final.

נימוק:

שאלה 13:

```
/* @inv: some invariant */
public class Base{
       /* @pre: i % 2 == 0 */
       /* @post: $ret != "" and $ret != null */
        */
       public String func(int i, int j){  /* some code here */}
   }

public class Sub extends Base{
   /* @pre: i % 2 == 0 && j > 0*/
   /* @post: $ret != null */
   public String func(int i, int j){ /* some code here */}
}
```

טענה 1 – Sub צריכה לקיים את כל השמורות (invariants) של Base, ויכולה להוסיף שמורות נוספות.

טענה 2 – תנאי ה pre של func ב Sub הוא תקין, על פי כללי ירושה ודריסה.

טענה 3 – תנאי ה post של func ב Sub הוא תקין, על פי כללי ירושה ודריסה.

בחר\י בתשובה הטובה ביותר:

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

נימוק:

public interface Map<K,V>

Modifier and Type	Method and Description
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.
boolean	containsValue(Object value) Returns true if this map maps one or more keys to the specified value.
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. Returns true if this set changed as a result of the call.
Boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this set if they're not already present. Returns true if this set changed as a result of the call.
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.

boolean	retainAll(Collection<?> c) Retains only the elements in this set that are contained in the specified collection.
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.
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
void	clear() Removes all of the elements from this list.
boolean	contains(Object o) Returns true if this list contains the specified element.
E	get(int index) Returns the element at the specified position in this list.
int	indexOf(Object o) 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>	iterator() Returns an iterator over the elements in this list in proper sequence.
boolean	remove(Object o) Removes the first occurrence of the specified element from this list, if it is present. Returns true if this list contained the specified element
boolean	retainAll(Collection<?> c) Retains only the elements in this list that are contained in the specified collection. Returns true if this list changed as a result of the call.
int	size() Returns the number of elements in this list.
default void	sort(Comparator<? super E> c) Sorts this list according to the order induced by the specified Comparator.

public final class String

char	charAt(int index) Returns the char value at the specified index.
int	compareTo(String anotherString) Compares two strings lexicographically.
static String	join(CharSequence delimiter, CharSequence... elements) Returns a new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter.
int	length() Returns the length of this string.
String[]	split(String regex) Splits this string around matches of the given regular expression.
String	trim() Returns a string whose value is this string, with any leading and trailing whitespace removed.