

גרסה מספר: \_\_\_\_\_ מספר סידורי: \_\_\_\_\_ מספר ת"ז: \_\_\_\_\_

## בחינה בתוכנה 1

סמסטר א' תשע"ח, מועד ב', 12 באפריל 2019  
לנה דנקין, ברית יונגמן, שי גרשטיין

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.

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.

יש לסמן את התשובה הטובה ביותר בתשובון. לא יינתן ניקוד על סימון תשובה בטופס הבחינה, במחברת הבחינה, או בטופס הנימוקים.

יש לנמק בתמצות את כל התשובות בטופס הנימוקים המצורף בלבד. נימוק חסר או לא נכון עלול לגרום לאי קבלת נקודות על שאלה במקרים שבהם יוחלט לקבל יותר מתשובה אחת. נימוק לתשובה אחרת מזו שסומנה על גבי טופס התשובות לא יתקבל.

יש לצרף את טופס המבחן למחברת הבחינה. מחברת ללא טופס עזר תפסל. תשובות ונימוקים במחברת הבחינה לא יבדקו.

יש למלא מספר סידורי (מספר מחברת) ומספר ת"ז על כל דף של טופס הבחינה.

אסור השימוש בחומר עזר כלשהו, כולל מחשבוניו או כל מכשיר אחר פרט לעט.

לטופס הבחינה מצורף דף לשאלות הסטודנטים. יש לכתוב שאלות שמתעוררות במהלך הבחינה בדף זה ולהעביר לסגל הקורס. שאלות ענייניות תענינה על ידי סגל הקורס בפני כל הנבחנות/ים.

## בהצלחה

© 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.

שאלות 1-5 מתייחסות למחלקה הבאה:

```
/*
@imp_inv: itemsList.size() == itemsSet.size();           //1
@inv: getSize() == itemsList.size()                     //2
@inv: nextInQueue() == null iff isEmpty() == true      //3
*/
public class MyUniquesQueue<T> {
    private final List<T> itemsList = new ArrayList<>();
    private final Set<T> itemsSet = new HashSet<>();

    /**
     * @pre !isEmpty()
     */
    public T dequeue() {
        T lastItem = itemsList.get(0);
        itemsSet.remove(lastItem);
        itemsList.remove(0);
        return lastItem;
    }

    public boolean enqueue(T item) {
        if (itemsSet.contains(item)) {
            return false;
        }
        itemsSet.add(item);
        itemsList.add(item);
        return true;
    }

    public Object nextInQueue() {
        if (isEmpty()) {
            return null;
        }
        return itemsList.get(0);
    }

    public int getSize() {
        return itemsList.size();
    }

    /** @post @return == true iff getSize() == 0 */
    public boolean isEmpty() {
        return itemsList.size() == 0;
    }
}
```

1. שאלה 1:

למחלקה MyUniquesQueue יש שלושה משתמרים – הראשון הוא משתמש ייצוג (implementation invariant) והשניים האחרים הם משתמרים רגילים (invariant). אילו מבין המשתמרים מנוסחים נכון על פי עקרונות כתיבת חוזים וגם תואמים למימוש המחלקה? בחר/י בתשובה הטובה ביותר:

- א. רק משתמר 1.
- ב. רק משתמר 2.
- ג. רק משתמר 3.
- ד. רק משתמרים 1+2.
- ה. רק משתמרים 1+3.
- ו. רק משתמרים 2+3.
- ז. כל המשתמרים.
- ח. אף אחד מבין המשתמרים.

2. שאלה 2:

נרצה להוסיף את השירות reorder אשר מסדר את האיברים של התור בסדר עולה לפי הסדר הטבעי שלהם.

```
public class MyUniquesQueue<T> implements Iterable<T>{
    private final List<T> itemsList = new ArrayList<>();
    private final Set<T> itemsSet = new HashSet<>();

    public void reodrder() {
        Collections.sort(itemsList);
    }

    /*
    public void reodrder2(Comparator<T> cmp) {
        Collections.sort(itemsList, cmp);
    } */

    // the rest of the code
}
```

בחר/י בתשובה הטובה ביותר:

- א. השירות reorder מתקמפל כמו שהוא, ואין צורך לבצע שינויים בקוד.
- ב. הקוד יתקמפל אם נשנה את הגדרת המחלקה ל `MyUniquesQueue<T> extends Comparable<T>>`
- ג. הקוד יתקמפל אם נשנה את הגדרת המחלקה ל `MyUniquesQueue<T> extends Comparable<T>`
- ד. לא ניתן לממש את MyUniquesQueue כך ש reodrder תתקמפל. הדרך היחידה לסדר את התור היא באמצעות שימוש בפונקציה reodrder2 שנמצאת בהערה, על פי הסדר שמגדיר ה Comparator שמתקבל כפרמטר, והסידור יבוצע לפי הסדר שמגדיר cmp.

## 3. שאלה 3:

נוסיף שירות אשר מפצל את התור לשני תורים:

```
public class MyUniquesQueue<T> {
    private final List<T> itemsList = new ArrayList<>();
    private final Set<T> itemsSet = new HashSet<>();

    public MyUniquesQueue<T> splitQueue() {
        MyUniquesQueue<T> newQueue = new MyUniquesQueue<>();
        List<Integer> toRemove = new ArrayList<>()
        for (int i = 0; i < itemsList.size(); i += 2) {
            newQueue.enqueue(itemsList.get(i));
            toRemove.add(i);
        }
        for (Integer indexToRemove : toRemove) {
            // removes element in index i
            itemsList.remove(indexToRemove.intValue());
        }
        return newQueue;
    }
    // the rest of the code
}
```

התבוננו בפונקציה main הבאה:

```
public static void main(String[] args) {
    MyUniquesQueue<Integer> q1 = new MyUniquesQueue<>();
    // some code that adds elements to q1
    MyUniquesQueue<Integer> q2 = q1.splitQueue();
    // *****
}
```

בחר/י בתשובה הטובה ביותר:

- הקוד המסומן ב \*\*\*\* יכול לייצר מצב שבו q1 מכיל איברים כפולים.
- הקוד המסומן ב \*\*\*\* יכול לייצר מצב שבו q2 מכיל איברים כפולים.
- הקוד המסומן ב \*\*\*\* יכול לייצר מצב הכנסת איבר ל q1 תיכשל למרות שהוא לא מופיע ב q1.
- הקוד המסומן ב \*\*\*\* יכול לייצר מצב הכנסת איבר ל q2 תיכשל למרות שהוא לא מופיע ב q2.
- מלבד תשובה זו, כל התשובות לא נכונות.
- מלבד תשובה זו, יש יותר מתשובה נכונה אחת.

## 4. שאלה 4:

נממש את הפונקציה iterator כך שתחזיר איטרטור אשר עובר רק על numOfItems האיברים הראשונים של התור

```
public class MyUniquesQueue<T> {
    private final List<T> itemsList = new ArrayList<>();
    private final Set<T> itemsSet = new HashSet<>();

    /* @pre numOfItems > 0 */
    public Iterator<T> iterator(int numOfItems) { /**
        return new Iterator<T>() {
            Iterator<T> it = itemsList.iterator();
            int currNum;

            @Override
            public boolean hasNext() {
                return currNum < numOfItems;
            }

            @Override
            public T next() {
                currNum++; /**
                return it.next(); /****
            }
        };
    }
    // the rest of the code
}

public class Test{
    public static void main(String[] args) {
        MyUniquesQueue<Integer> q = new MyUniquesQueue<>();
        int numIfItems; /*will be initialized later*/;
        // some code here, including the initialization of numOfItems
        Iterator<Integer> qIt = q.iterator(numIfItems);
        while(qIt.hasNext()) {
            System.out.println(qIt.next()); /**
        }
    }
}
```

בפונקציה main של Test חלק מהקוד לא נתון. בקוד המוסתר אין שגיאות קומפילציה/זמן ריצה, והוא לא מבצע השמה של q ל null.

בחרי' בתשובה הטובה ביותר:

א. השירות iterator מתקמפל. בהרצת התוכנית Test האיטרטור יעבור על numOfItems האיברים הראשונים של q כנדרש.

ב. ישנה שגיאת קומפילציה בשורה המסומנת ב \*

ג. ישנה שגיאת קומפילציה בשורה המסומנת ב \*\*

ד. ישנה שגיאת קומפילציה בשורה המסומנת ב \*\*\*

ה. השירות iterator מתקמפל. בהרצת התוכנית Test האיטרטור qIt עלול לזרוק שגיאה בזמן ריצה בשורה המסומנת ב #.

ו. השירות iterator מתקמפל. בהרצת התוכנית Test הלולאה על qIt יכולה לעבור על מספר איברים הגדול מ numOfItems.

5. שאלה 5:

שאלה זו מתייחסת למימוש ההתחלתי של MyUniquesQueue כפי שהוצג בשאלה 1. נרצה לבצע שינוי בהתנהגות של המחלקה MyUniquesQueue.

טענה 1: על ידי שינוי שורת קוד אחת בלבד בפונקציה enqueue ניתן להפוך את התור למחסנית.

טענה 2: ניתן לוותר על מבנה הנתונים itemsSet ולשמר על ההתנהגות של המחלקה, אך זה יכול להגדיל את זמן הריצה.

טענה 3: ניתן לוותר על הקוד אשר מונע הכנסת איברים כפולים באמצעות שינוי החוזה של enqueue כך שלא יאפשר הכנסת איברים כפולים. פתרון זה שקול לפתרון הממומש מבחינת אופן השימוש של הלקוח במחלקה MyUniquesQueue.

בחר/י בתשובה הטובה ביותר:

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

6. שאלה 6:

```
public class Test {

    public static interface I<T>{
        public int func(T i, T j); //$
    }

    public static int func(I i1, I i2, I i3) {
        return i3.func(i1, i2); //#
    }

    public static void main(String[] args) {
        I<Integer> i1 = (x,y) -> (x+1)*y;
        I<Integer> i2 = (x,y) -> x*y*2;
        I<I<Integer>> i3 = (x,y) -> Math.max(x.func(3, 4), y.func(1, 5));
        System.out.println(func(i1, i2, i3)); //&
    }
}
```

בחר/י בתשובה הטובה ביותר:

- א. קיימת שגיאת קומפילציה בשורה המסומנת ב \$
- ב. קיימת שגיאת קומפילציה בשורה המסומנת ב #
- ג. קיימת שגיאת קומפילציה בשורה המסומנת ב &
- ד. התוכנית מתקמפלת אך בזמן ריצה נזרק חריג מסוג ClassCastException בשורה המסומנת ב #
- ה. התוכנית מתקמפלת ומדפיסה 16

```
public class Test {
    public static void main(String[] args) {
        Stream<Integer> s= Stream.generate(new NaturalNumbers());
        boolean res = s.filter(x-> x %10 == 0)
            .allMatch(x -> {
                System.out.print("!");
                return x%25 == 0;
            });
        System.out.println(res);
    }
}

public class NaturalNumbers implements Supplier<Integer>{
    private int i = 0;
    private static final int MAX_VALUE = 100;

    public Integer get() {
        i = (i + 1) % MAX_VALUE;
        return i;
    }
}
```

תזכורת: השירות allMatch פועל בדומה לשירות anyMatch. להלן התיעוד שלו:

`boolean allMatch(Predicate<? super T> predicate)`

Returns whether all elements of this stream match the provided predicate.

בחר/י בתשובה הטובה ביותר:

- א. התוכנית אינה מבצעת שום הדפסה ואינה עוצרת.
- ב. התוכנית מדפיסה `false` ומסתיימת.
- ג. התוכנית מדפיסה `true` ומסתיימת.
- ד. התוכנית מדפיסה אינסוף פעמים את הסימן ! ולא עוצרת.
- ה. כל התשובות מלבד זו לא נכונות.
- ו. התוכנית מדפיסה רק `false` ומסתיימת.
- ז. התוכנית מדפיסה רק `true` ומסתיימת.

8. שאלה 8:

```

public class Box<V> {
    public void func1(List<?> lst1, List<?> lst2) { /*some code here */}

    public void func2(List lst1, List lst2) { /*some code here */}

    public void func3(List<V> lst1, List<V> lst2) { /*some code here */}

    public <T> void func4(List<T> lst1, List<T> lst2) { /*some code here */}

    public static void main(String[] args) {
        //box, lst1, lst2 are defined here.
        box.func1(lst1, lst2);
    }
}

```

לפניכם הקוד של המחלקה Box. הפונקציה main מכילה קוד שחלקו לא נתון, אך ידוע שהוא מתקמפל ורץ.

- טענה 1: אם נחליף את הקריאה ל func1 ב func2, הקוד ימשיך להתקמפל בוודאות.  
טענה 2: אם נחליף את הקריאה ל func1 ב func3, הקוד ימשיך להתקמפל בוודאות.  
טענה 3: אם נחליף את הקריאה ל func1 ב func4, הקוד ימשיך להתקמפל בוודאות.

בחר/י בתשובה הטובה ביותר:

- כל הטענות נכונות.
- כל הטענות לא נכונות.
- רק טענה 1 נכונה.
- רק טענה 2 נכונה.
- רק טענה 3 נכונה.
- רק טענות 1+2 נכונות.
- רק טענות 1+3 נכונות.
- רק טענות 2+3 נכונות.

9. שאלה 9:

מהי הטענה הנכונה? בחר/י את התשובה הנכונה:

- שדות סטטיים (static members) נשמרים על ה stack.
- כאשר מריצים תוכנית MyProg כלשהי, השדות הסטטיים של כל המחלקות האחרות בפרוייקט שבו MyProg נמצאת נוצרים בזכרון בתחילת הריצה של התוכנית.
- שדות מופע (members) נשמרים על ה stack.
- משתנים מקומיים אשר מצביעים לטיפוסי הפניה (references) נוצרים על ה heap.
- שדות מופע וסטטיים מקבלים ערכים דיפולטיים.
- מלבד תשובה זו, יש יותר מתשובה אחת נכונה.



```
public class Test<V> {

    public static <V> boolean f1(V v1, V v2) {
        return v1.equals(v2);
    }

    public boolean f2(List<? extends V> l) {
        return f1(l.get(0), l.get(1));
    }

    public static void main(String[] args) {
        Test<String> b = new Test<>();
        *****
    }
}
```

לפניכם מספר אופציות בהן נוכל להחליף את \*\*\*\*\*

- אופציה 1: `b.f1(1, 3);`
- אופציה 2: `b.f2(Arrays.asList(1,3));`
- אופציה 3: `b.f2(Arrays.asList("1","3"));`

עבור אילו מבין האופציות הקוד של Test יתקמפל (הניחו שכל אחת מהפונקציות מתווספת בנפרד)?

- א. אופציה 1 בלבד.
- ב. אופציה 2 בלבד.
- ג. אופציה 3 בלבד.
- ד. אופציות 1+2 בלבד.
- ה. אופציות 1+3 בלבד.
- ו. אופציות 2+3 בלבד.
- ז. עבור כל האופציות.
- ח. עבור אף אחת מבין האופציות.

11. שאלה 11:

```
public class A {
    protected A f(int x, int y) throws IOException {
        return new A();
    }
}

public class C extends A {}

public class B extends A {
    // A f(int x, int y) throws EOFException { return new A(); }

    // protected A f(int x, int y) throws Exception { return new A(); }

    // public B f(int x, int y) throws EOFException { return new B(); }

    // A f(double x, double y) throws Exception { return new A(); }

    // protected C f(int x, int y) throws IOException { return new C(); }
}
```

מחלקות A, B ו-C נמצאות באותה חבילה. הניחו כי בכל פעם נוסף מתודה אחת בלבד למחלקה B. בחרו בתשובה הנכונה ביותר.

- א. עבור כל מתודה שנוסף הקוד יתקמפל.
- ב. עבור כל מתודה שנוסף תיווצר שגיאת קומפילציה.
- ג. ישנה בדיוק מתודה אחת שהוספתה תגרור שגיאת קומפילציה.
- ד. ישנן בדיוק שתי מתודות שהוספתן תגרור שגיאת קומפילציה.
- ה. ישנן בדיוק שלוש מתודות שהוספתן תגרור שגיאת קומפילציה.
- ו. ישנן בדיוק ארבע מתודות שהוספתן תגרור שגיאת קומפילציה.

12. שאלה 12

לפניכם מספר טענות על המילה השמורה final. חלק מהסעיפים מתייחסים לקוד של המחלקה B.

```
public class B {
    private final List<Integer> a;

    public B() { setA(); }
    public void setA() { a = Arrays.asList(1,2,3); }
}
```

- א. אם משתנה x כלשהו מצביע לאובייקט שהוא immutable, אין הבדל הבדל מבחינת קומפילציה והתנהגות הקוד בזמן ריצה בין אם היינו מגדירים אותו כ final ובין אם לא.
- ב. הקוד של המחלקה B מתקמפל.
- ג. הקוד של המחלקה B לא מתקמפל, הוא יתקמפל אם setA תוגדר להיות final.
- ד. הקוד של המחלקה B לא מתקמפל, הוא יתקמפל אם a יוגדר להיות static.
- ה. מלבד תשובה זו יש יותר מתשובה אחת נכונה.
- ו. מלבד תשובה זו, כל התשובות לא נכונות.

13. שאלה 13:

שאלה זו מתייחסת למימוש של מחלקות A, B שמימושן אינו נתון. ניתן להניח את קיומן של מחלקות נוספות ללא שום מגבלות.

```
public void func(B b1, B b2) {  
    A a1 = (A) b1; // line 1  
    A a2 = (A) b2; // line 2  
    A a3 = b2; // line 3  
}
```

להלן מספר טענות לגבי קוד זה:

- טענה 1: ניתן לממש את A ו-B כך שרק אחת משלוש השורות תתקמפל.
- טענה 2: ניתן לממש את A ו-B כך שרק שורות 1 ו-2 תתקמפלנה.
- טענה 3: ניתן לממש את A ו-B כך ששלוש השורות תתקמפלנה ועלולה להיזרק שגיאת זמן ריצה.

בחר/י את התשובה הנכונה ביותר:

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

14. שאלה 14

```
public class A {  
  
    public A() {}  
  
    public A(int i) {  
        this();// ***  
        System.out.print("A.i=" + i + " ");  
    }  
}  
  
public class B extends A {  
  
    public B() { this(0); } // *  
  
    public B(int i) { System.out.print("B.i=" + i + " "); } // **  
  
    public static void main(String[] args) {  
        B b = new B();  
    }  
}
```

מה יקרה בהרצת התוכנית B? בחר/י בתשובה הנכונה ביותר:

- א. הקוד לא יתקמפל, שגיאת קומפילציה בשורה המסומנת ב-\*
- ב. הקוד לא יתקמפל, שגיאת קומפילציה בשורה המסומנת ב-\*\*
- ג. הקוד לא יתקמפל, שגיאת קומפילציה בשורה המסומנת ב-\*\*\*
- ד. ישנה יותר משגיאת קומפילציה אחת
- ה. **הקוד יתקמפל וידפיס B.i=0**
- ו. הקוד יתקמפל וידפיס A.i=0 B.i=0

15. שאלה 15:

להלן מספר טענות על מחלקות אבסטרקטיות:

- טענה 1: מחלקה אבסטרקטית יכולה להכיל בנאי פרטי.
- טענה 2: מחלקה אבסטרקטית יכולה להכיל שדה פרטי.
- טענה 3: מחלקה אבסטרקטית יכולה להכיל שירות אבסטרקטי פרטי.

בחר/י בתשובה הטובה ביותר:

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

16. שאלה 16:

```
public class class A {
    private String s = "A";

    public A() {
        System.out.print(this.s + " ");
        foo();
    }

    //Q17 - remove static
    public static void foo() { System.out.print("A.foo() "); }
}

public class B extends A {

    private String s = "B";

    public B() {
        System.out.print(this.s + " ");
        foo();
    }

    //Q17 - remove static
    public static void foo() { System.out.print("B.foo() "); }

    public static void main(String[] args) {
        A a = new B();
    }
}
```

מה יודפס בהרצת התוכנית B?

- א. A A.foo() B B.foo()
- ב. null A.foo() B B.foo()
- ג. null B.foo() B B.foo()
- ד. B A.foo() B B.foo()
- ה. B B.foo() B B.foo()
- ו. A A.foo() A A.foo()
- ז. A B.foo() B B.foo()
- ח. B A.foo() B A.foo()

17. שאלה 17:

השאלה מתייחסת לקוד אשר הוצג בשאלה 16. כיצד תשתנה התשובה אם נשנה את המתודה foo בשתי המחלקות, A ו-B כך שתהיה מתודת מופע לא מתודה סטטית?

- א. התשובה תשתנה וכעת יודפס: A B.foo() B B.foo()
- ב. התשובה תשתנה וכעת יודפס: null B.foo() B B.foo()
- ג. התשובה תשתנה וכעת יודפס: B B.foo() B B.foo()
- ד. התשובה תשתנה וכעת יודפס: A A.foo() B B.foo()
- ה. התשובה תשתנה וכעת יודפס: null B.foo() A B.foo()
- ו. התשובה תשתנה וכעת יודפס: A B.foo() A B.foo()

```
public interface I { }

public class A implements I { }

public class B extends A {
    public void foo() {}
}

public class C extends A {
    public static void foo(/*****/) {
        ((B)x).foo();
    }

    public static void main(String[] args) {
        foo(new A());
        foo(new B());
        foo(new C());
    }
}
```

נרצה להשלים את ה-\*\*\*\*\* באחת מהאפשרויות הבאות, כך שהתכנית תתקמפל ותרוץ ללא שגיאות זמן ריצה. אילו מבין האופציות מתאימה? בחר/י בתשובה הטובה ביותר.

- א. נחליף את /\*\*\*\*\*/ ב- x I
- ב. נחליף את /\*\*\*\*\*/ ב- x A
- ג. נחליף את /\*\*\*\*\*/ ב- x B
- ד. נחליף את /\*\*\*\*\*/ ב- x Object
- ה. נחליף את חתימת המתודה של foo ב- `public static <T> void foo(T x)`
- ו. כל התשובות מלבד תשובה זו לא נכונות.
- ז. מלבד תשובה זו, יש יותר מתשובה אחת נכונה.

19. שאלה 19:

```
public class A {
    public int i = 1;
    public static int j = 2;

    // in Q-20, change to public static class B
    public class B extends A{
        public void foo() {
            System.out.print(this.i); /**
            System.out.print(j); /**#
        }
    }
    public static void main(String[] args) {
        B b = new B(); /**%
        b.foo(); /**$
    }
}
```

בחר/י בתשובה הטובה ביותר:

- א. ישנה שגיאת קומפילציה אחת בלבד, והיא נמצאת בשורה המסומנת ב-\*
- ב. ישנה שגיאת קומפילציה אחת בלבד, והיא נמצאת בשורה המסומנת ב-#
- ג. ישנה שגיאת קומפילציה אחת בלבד, והיא נמצאת בשורה המסומנת ב-\$
- ד. ישנה שגיאת קומפילציה אחת בלבד, והיא נמצאת בשורה המסומנת ב-%
- ה. ישנה יותר משגיאת קומפילציה אחת
- ו. הקוד מתקמפל ומדפיס: 1 2 3 (בשורות נפרדות)

20. שאלה 20:

נשנה את הקוד שמופיע בשאלה 19 כך ש B תוגדר באופן הבא:

```
public static class B{
    //the rest of the code is the same
}
```

כעת, מה תהיה תשובתך לשאלה הקודמת:

- א. ישנה שגיאת קומפילציה אחת בלבד, והיא נמצאת בשורה המסומנת ב-\*
- ב. ישנה שגיאת קומפילציה אחת בלבד, והיא נמצאת בשורה המסומנת ב-#
- ג. ישנה שגיאת קומפילציה אחת בלבד, והיא נמצאת בשורה המסומנת ב-\$
- ד. ישנה שגיאת קומפילציה אחת בלבד, והיא נמצאת בשורה המסומנת ב-%
- ה. ישנה יותר משגיאת קומפילציה אחת.
- ו. הקוד מתקמפל ומדפיס: 1 2 3 (בשורות נפרדות)

```
public class Test {
    public static void main(String[] args) {
        int x = 0;
        String[] arr = { "a", "ab", null};
        for (String s : arr) {
            try {
                if (s.length() % 2 == 0) {
                    throw new IOException();
                }
            }
            catch(IOException exp) {
                x++;
                throw new RuntimeException();
            }
            catch(RuntimeException exp) {
                x++;
            }
            finally{
                x++;
                System.out.print(x + " ");
            }
            System.out.print(x + " ");
        }
        System.out.println(x);
    }
}
```

מה יקרה בהרצת הקוד הבא?

- א. יודפס 1 1 3 ולאחר מכן התוכנית תעוף בגלל שגיאה.
- ב. יודפס 1 ולאחר מכן התוכנית תעוף בגלל שגיאה.
- ג. יודפס 1 3 ולאחר מכן התוכנית תעוף בגלל שגיאה.
- ד. יודפס 1 1 3 3 5 ולאחר מכן התוכנית תעוף בגלל שגיאה.
- ה. יודפס 1 1 3 3 5 5! והתוכנית תסתיים ללא שגיאות. במסית זה צריך להתעלם מסימן הקריאה
- ו. יודפס 1 1 4 והתוכנית תעוף בגלל שגיאת זמן ריצה.
- ז. יודפס 1 1 4 4 6 6! והתוכנית תסתיים ללא שגיאות. במסית זה צריך להתעלם מסימן הקריאה
- ח. יודפס 1 1 4 4 6 ולאחר מכן התוכנית תעוף בגלל שגיאה.