

בחינה בתוכנה 1

סמסטר א תשפ"ג, מועד א', 30 בינואר 2023
לנה דנקין, אלה גולדשמידט, אמיר ברדה

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:

1. הקוד שמופיע במבחן מתאים לגרסה 17 Java.
2. כל החבילות הדרושות יובאו, ואין צורך לכתוב שורות import בגוף הקוד.
3. כל מחלקה שהיא public מופיעה בקובץ Java משלה.
4. בכל שאלה, כל המחלקות מופיעות באותה חבילה (package).
5. 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.
6. בסוף הבחינה מופיע נספח עם תיעוד של מחלקות שאתם עשויים לעשות בהן שימוש בחלק הפתוח של הבחינה.
7. הקוד שאתם נדרשים לספק צריך להיות יעיל ולהימנע ממחזור קוד. חלק מהציון ניתן גם היבטים אלה, ולא רק על נכונות הפתרון.

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.

מבנה הבחינה:

- הבחינה מורכבת משני חלקים: חלק פתוח (שתי שאלות על סך 55 נקודות) ושאלות אמריקאיות (10 שאלות, כל אחת שווה 5 נק'). עליכם לענות על הבחינה באופן הבא:
1. בשאלות הפתוחות להשלים את הקוד החסר במקומות המסומנים ע"י מסגרת. שימו לב שלא חייבים למלא את כל המסגרות.
 2. 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!

דוגמא להוספת צוללות ללוח משחק בגודל 6x6 (ע"י קריאה לאחד הבנאים שתממשו בהמשך):

```
IBattleshipBoard board = new /**some constructor**(6, 6);
board.addBattleship(3,0, Direction.DOWN, 2);
board.addBattleship(1,2, Direction.DIAG, 3);
```

0						
1						
2						
3						
4						
5						

התרשים שמופיע מצד שמאל מתאר לוח המשחק שמתקבל. התאים התפוסים ע"י שתי הצוללות מושחרים.

הצוללת הראשונה שמתווספת היא צוללת באורך 2 שתופסת את התאים <3,0> ו <4,0> (נשתמש בסימון <x,y> בשביל להתייחס לשורה x ועמודה y).

הצוללת השניה שמתווספת היא צוללת באורך 3. היא מתחילה מהתא <1,2> ומסתיימת בתא <3,4> (בשונה ממשחק הצוללות המוכר, אנחנו מאפשרים ציור צוללות גם באלכסון).

בכל הוספת צוללת יש לשים לב לכך שהצוללת לא חולקת מיקום עם צוללת אחרת ולא סמוכה לשום צוללת אחרת (גם לא באלכסון). ניתן להניח שהצוללת לא חורגת מתחום הלוח, ואין צורך לבדוק את זה. לדוגמא: הקריאה addBattleship(5,5, Direction.RIGHT, 5) אינה חוקית.

0		X	X	X		
1		X		X	X	
2	X	X	X		X	X
3		X	X	X		X
4		X		X	X	X
5	X	X				

כל המקומות שבהם לא יכולה לעבור צוללת מסומנים ב X-ים בתרשים השני.

- לדוגמא, בלוח המופיע בדוגמא, לא ניתן להוסיף את הספינות הבאות:
 - א. <3,0>, DOWN, 1 – הצוללת חולקת מיקום עם צוללת קיימת (התא המשותף הוא <3,0>).
 - ב. <4,5>, DOWN, 2 – הצוללת סמוכה לצוללת קיימת (התא <4,5> סמוך לתא <3,4> השייך לצוללת קיימת).

לרשותכם מחלקה אבסטרקטית ממנה ירשו שתי המחלקות אותן תממשו באופן חלקי בשאלה זו:

```
public abstract class AbsBattleshipBoard implements IBattleshipBoard {
    protected int rowsNum, columnsNum;

    public int getRowsNum () { return rowsNum;}

    public int getColumnsNum() { return columnsNum;}
}
```

את המנשק של IBattleshipBoard תממשו בשני אופנים:

המחלקה ArrBattleshipBoard:

מחלקה זו מייצגת את הלוח ע"י מערך דו מימדי בוליאני בגודל לוח המשחק. במחלקה שדה יחיד בשם boardArr כך שמתקיים: boardArr[i][j] == true אם ורק אם התא שנמצא בשורה i ועמודה j תפוס ע"י צוללת כלשהי.

זהו שלד המחלקה החלקי המכיל את המימושים שסופקו עבורכם.

```
public class ArrBattleshipBoard extends AbsBattleshipBoard {

    private boolean[][] boardArr;

    @Override
    public boolean isOccupied(int row, int col) {
        return boardArr[row][col];
    }
}
```

עמוד 4 מתוך 20

המחלקה MapBattleshipBoard:

מחלקה זו מייצגת את לוח המשחק ע"י 2 מפות (Map), אחת עבור השורות (rows), ואחת עבור העמודות (columns). המפתחות ב rows הם האינדקסים השורות, והערך הוא קבוצה (Set) אשר מכילה את האינדקסים של כל העמודות התפוסות באותה השורה. כנ"ל לגבי ה columns. לדוגמא, עבור הלוח מהדוגמא הקודמת:

```
rows = { 1: {2}, 2: {3}, 3: {0, 4}, 4: {0} }  
columns = { 0: {3,4}, 2: {1}, 3: {2}, 4: {3} }
```

הסבר: השורות שבהן מופיעות צוללות הן: 1,2,3,4 ולכן אלה הם המפתחות של rows. העמודות שבהן מופיעות צוללות הן 0,2,3,4, אלה הם המפתחות של columns.

```
/* @imp_inv: rows.containsKey(row) if !rows.get(row).isEmpty()  
 * @imp_inv: columns.containsKey(col) if !columns.get(col).isEmpty()  
 * @imp_inv: rows.get(row).contains(col) if and only if  
 * columns.get(col).contains(row) */  
public class MapBattleshipBoard extends AbsBattleshipBoard {  
    private Map<Integer, Set<Integer>> rows;  
    private Map<Integer, Set<Integer>> columns;  
  
    @Override  
    public boolean isOccupied(int row, int col) {  
        return rows.containsKey(row) && rows.get(row).contains(col);  
    }  
}
```

סעיף א' (7 נק')

ממשו את הבנאים של שתי המחלקות ArrBattleshipBoard ושל MapBattleshipBoard. אם קיים קוד משותף לשני הבנאים, ניתן לעדכן גם את המחלקה האבסטרקטית. כמו כן, ניתן להוסיף שירותי עזר לכל המחלקות. אין להוסיף שדות נוספים למחלקות.

ArrBattleshipBoard:

```
/* @post: for each <i,j> on the board, isOccupied(i,j) == False*/  
public ArrBattleshipBoard(int rowsNum, int columnsNum) {
```

```
    super(rowsNum, columnsNum);  
    this.boardArr = new boolean[rowsNum][columnsNum];
```

```
}
```

MapBattleshipBoard:

```
/* @post: for each <i,j> on the board, isOccupied(i,j) == False*/  
public MapBattleshipBoard(int rowsNum, int columnsNum) {
```

```
    super(rowsNum, columnsNum);  
    this.rows = new HashMap<>();  
    this.columns = new HashMap<>();
```

```
}
```

```
//other code
in AbsBattleshipBoard:
public AbsBattleshipBoard(int rowsNum, int columnsNum){
    this.rowsNum = rowsNum;
    this.columnsNum = columnsNum;
}
```

סעיף ב' (6 נק'): _____

עבור מימוש השירות addBattleship, נוסיף שירות עזר במחלקה AbsBattleshipBoard. שם השירות הוא getBattleshipCells והוא יחזיר את רשימת כל התאים שאמורה לתפוס הצוללת המיוצגת ע"י קלט הכולל שורה, עמודה, כיוון ואורך. סדר התאים ברשימה שתוחזר אינו משנה. וודאו שעבור ספינה באורך length יחזרו בדיוק length איברים ברשימה.

נשתמש במחלקה Location בשביל לייצג את המיקומים המוחזרים. המחלקה ממומשת כך:

```
public record Location(int row, int col) {}
```

(תזכורת לשימוש ב record ניתן למצוא בנספח המופיע בסוף הבחינה).

לדוגמא, הקריאה: `getBattleshipCells(3, 0, Direction.DOWN, 2)`

תחזיר רשימה עם שני איברים מטיפוס Location המייצגים את המיקומים `<3,0>`, `<4,0>`

```
protected List<Location> getBattleshipCells(int row, int col,
                                          Direction dir, int length){
```

```
List<Location> cells = new ArrayList<>();
for (int i = 0; i < length; i++) {
    int currRow = row+i*dir.getDRows();
    int currCol = col+i*dir.getDCols();
    cells.add(new Location(currRow, currCol));
}
return cells;
```

```
}
```

סעיף ג' (9 נק'): _____

ממשו את השירות addBattleship של המחלקה ArrBattleshipBoard. אין צורך להשלים את המימוש עבור MapBattleshipBoard, אך עליכם לקחת בחשבון את קיומה של מחלקה זו (גם אם מימושה לא נדרש) ולהחליט אם קיים קוד משותף בין שני המימושים. אם קיים קוד משותף, הוסיפו אותו למחלקה האבסטרקטית. כמו כן, ניתן להוסיף שירותי עזר למחלקות.

הנחיה: מומלץ לבחון בנפרד כל אחד מהתאים אותם אמורה לתפוס הצוללת (גם אם בפועל חלק מהמיקומים נבדקים יותר מפעם אחת).

```
//In AbsBattleshipBoard
public boolean addBattleship(int row, int col,
                             Direction dir, int length) {

    var battleshipCells = getBattleshipCells(row, col, dir, length);
    //check if OK to add
    for (Location currCell : battleshipCells) {
        if (! checkSquareIsFree(currCell.row(), currCell.col())) {
            return false;
        }
    }
}
```

```

//add battleship
for (Location currCell : battleshipCells) {
    markAsOccupied(currCell.row(), currCell.col());
}
return true;
}

private boolean checkSquareIsFree(int row, int col) {
    for (int i = row-1; i <= row+1; i++){
        for (int j = col-1; j <= col+1; j++) {
            if (i >= 0 && i < this.getRowsNum() &&
                j >= 0 && j < this.getColumnsNum()) {
                if (isOccupied(i, j)) {
                    return false;
                }
            }
        }
    }
    return true;
}

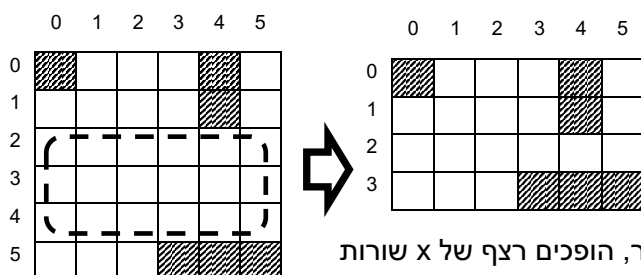
protected abstract void markAsOccupied(int row, int col);

//In ArrBattleshipBoard
protected void markAsOccupied(int row, int col) {
    this.boardArr[row][col] = true;
}

```

סעיף ד' (7 נק')::

בסעיפים ד' ו ה' תממשו את השירות squeezeRows אשר מכווצת (אם ניתן) את רצף השורות הריקות הארוך ביותר בלוח. הלוח מצד שמאל הוא הלוח המקורי, והלוח מצד ימין הוא הלוח המכווץ.



- (1) נזהה את כל השורות הריקות בלוח.
- (2) נזהה את רצף השורות הריקות הארוך ביותר (מסומן במרובע מקווקו בתרשים השמאלי).
- (3) אם אורך הרצף הארוך ביותר הוא x ו $x > 1$, נכווץ את הלוח ע"י מחיקת $x-1$ שורות מתוך רצף השורות הריקות (כלומר, הופכים רצף של x שורות ריקות לשורה ריקה אחת)

בדוגמא המצורפת, מספר השורות בלוח קטן, וגודל הלוח השתנה להיות 4×6 .

את הלוח שמופיע בעמוד 3 לא ניתן לכווץ (רצף השורות הריקות הארוך ביותר הוא בגודל 1).

בסעיפים ד' וה' תממשו את הכיוון **באופן חלקי**. את המימוש של שלב (2) סיפקנו לכם בפונקציית עזר שבה תוכלו להשתמש (פירוט בהמשך), כך שעליכם לטפל במימוש שלב (1) – זיהוי שורות ריקות, ושלב (3) - מחיקת שורות ריקות. חלק מהקוד יכול להיות משותף לשתי המחלקות, ולכן בסעיף ד' תממשו את החלקים המשותפים בתוך המחלקה האבסטרקטית, ובסעיף ה' תשלימו את המימוש של MapBattleShipBoard. **לא נדרש מימוש עבור ArrBattleShipBoard** (אך עליכם לקחת את קיומה של המחלקה הזו בחשבון בעת החלוקה לקוד משותף ולקוד נפרד).

מתודת עזר עבור שלב (2):

המתודה getSqueezeRowsParam מקבלת מערך בוליאני שאורכו כמספר השורות על לוח המשחק. אם בתא ה i מופיע הערך true, זה אומר שהשורה ה i היא ריקה.

הערה: המימוש של פונקציית העזר לא נתון, ועליכם להשתמש בה על פי ההסבר המסופק.

עבור הלוח המקורי בדוגמא שלנו, הייצוג ע"י מערך הוא:

```
rowsArr = [false, false, true, true, true, false]
```

הפונקציה מחזירה מפה (Map) עם 2 זוגות של מפתח/ערך: למפתח "startRow" ימופה אינדקס השורה שהחל ממנה מתחיל הרצף הריק הארוך ביותר. למפתח "deleteNum" ימופה מספר השורות למחיקה. עבור המערך rowsArr צריך למחוק 2 שורות החל משורה 2, ולכן הפונקציה תחזיר:

```
{ "deletedNum": 2, "startRow": 2, }
```

במילים אחרות, עליכם לייצר את המערך הבוליאני עבור שורות הלוח, לקרוא לפונקציה getSqueezeRowsParam ואז למחוק החל מהשורה "startRow" מספר של "deletedNum" שורות.

ממשו את השירות squeezeRows של המחלקה האבסטרקטית. תזכורת – הפונקציה מחזירה את מספר השורות שנמחקו. הוסיפו שירותי עזר כרצונכם:

```
public abstract class AbsBattleShipBoard implements IBattleShipBoard {
    /* previous code */
```

```
protected Map<String, Integer> getSqueezeRowsParam(
    boolean[] rowsVec){
    /* implementation is not provided */
}
```

```
/* @post: getRowsNum() == prev(getRowsNum()) - $ret */
public int squeezeRows() {
```

```
    boolean[] emptyRowsVec = new boolean[this.rowsNum];
    for (int i = 0 ; i < this.rowsNum; i++) {
        emptyRowsVec[i] = checkIsEmpty(i);
    }

    var squeezeParams = getSqueezeRowsParam(emptyRowsVec);

    /* also possible to split updateRows into two steps:
    1. delete rows from startRow till startRow+deletedNum
    (implementation is irrelevant for MapBattleShipBoard)
    2. update rows from startRow+deletedNum+1 till the end of the
    board */
    updateRows(squeezeParams.get("startRow"),
                squeezeParams.get("deletedNum"));
    this.rowsNum -= squeezeParams.get("deletedNum");
    return squeezeParams.get("deletedNum");
```

```
}
```

```
protected abstract boolean checkIsEmpty(int row);
protected abstract void updateRows(int from, int numRows);
```

סעיף ה' (8 נק')

השלימו את המימוש של squeezeRows במחלקה MapBattleshipBoard.

```
protected boolean checkIsEmpty(int row) {
    return !rows.containsKey(row);
}

protected void updateRows(int from, int toDelete) {
    /* the rows from "from" till "from"+"toDelete" are empty and
    going to be deleted. No need to delete them empty rows are not saved
    in the rows/columns. The row "from" + "toDelete" is the empty row that
    remains after the deletion, so we can start updating from "from" +
    "toDelete" + 1. It's also OK to start from "from", and let the loop
    run till it gets to the rows that need to be updated.
    Also, Note that is it important to iterate in an ascending order */
    for (int i = from+toDelete+1; i < this.rowsNum; i++) {
        if (rows.containsKey(i)) {
            //update rows
            int newI = i-toDelete;
            rows.put(newI, rows.get(i));
            rows.remove(i);

            //update columns
            for (int col :rows.get(newI)) {
                columns.get(col).remove(i);
                columns.get(col).add(newI);
            }
        }
    }
}
```


שאלה 2 (17 נק')

סעיף א' (9 נק'):

ממשו את הפונקציה analyzeText אשר מקבלת רשימה של מחרוזות המייצגות משפטים. התוים היחידים בכל מחרוזת הם אותיות בשפה האנגלית (גדולות וקטנות, רווחים וירידות שורה).

עליכם להדפיס לכל אות בשפה האנגלית שיש מילים אשר מתחילות בה את k המילים הכי שכיחות שמתחילות באות זו. עליכם להמיר את כל המילים לאותיות קטנות (lowercase) וכל הספירות והדפסות יהיו ב lowercase. האותיות יודפסו לפי סדר הא"ב, והמילים עבור כל אות יודפסו בסדר יורד לפי שכיחות על פי הפורמט הבא:

לדוגמא עבור הרשימה שמכילה את המשפטים הבאים:

Love Love me do drive my car
Do love my light lamp life

כאשר k=3 יודפס הפלט הבא:

c: car
d: do, drive
l: love, lamp, life
m: my, me

הנחיות:

- א. אם יש 2 מילים המופיעות אותו מספר הפעמים, עליכם לסדר אותן בסדר לקסיקוגרפי עולה.
- ב. יודפסו לכל היותר k מילים. אם לא קיימות k מילים המתחילות באות כלשהי, יודפסו פחות מ k, אך אם למשל k=2 ויש 3 מילים הכי שכיחות, עליכם לבחור את ה 2 הראשונות בסדר לקסיקוגרפי ולהדפיס רק אותן (זו הסיבה שבגללה המילה light לא הודפסה, למרות שהיא מופיעה פעם אחת, בדיוק כמו life ו lamp).

```

/* @pre: k > 0 */
public static void analyzeText(List<String> sentences, int k) {
    //build counts map
    Map<Character, Map<String, Integer>> counts =
        new HashMap<>();
    for (char c = 'a'; c <= 'z'; c++) {
        counts.put(c, new HashMap<>());
    }
    for (String sentence:sentences) {
        for(String word: sentence.split(" ")) {
            word = word.trim().toLowerCase();
            if (word.length() >0) {
                char firstChar = word.charAt(0);
                var cnt = counts.get(firstChar); //we know it's a Map
                cnt.put(word, cnt.getOrElseDefault(word, 0)+1);
            }
        }
    }
    //print up to k elements for each char
    for (char c = 'a'; c <= 'z'; c++) {
        if (!counts.get(c).isEmpty()) {
            var toPrint = counts.get(c).entrySet().stream()
                .sorted((x,y) -> {
                    //reversed counts order

```

```

        int cnt = -Integer.compare(x.getValue(), y.getValue());
        //lexicographic order in case counts are the same
        return cnt != 0 ? cnt : x.getKey().compareTo(y.getKey());
    })
    .map(x->x.getKey()) //leave only the words
    .limit(k) // up to k elements
    .collect(Collectors.toList());
System.out.println(c + " : " + String.join(", ",toPrint));
}
}
}

```

סעיף ב' (8 נק'):

ממשו את הפונקציה `getLongestArrProg` אשר מקבלת מערך של מספרים שלמים, ומדפיסה את אורך הסדרה החשבונית הרציפה הארוכה ביותר במערך, וכן את הפרש הסדרה (הניחו לשם פשטות שקיימת סדרה ארוכה ביותר). לדוגמא, עבור המערך `[1, 4, 3, 5, 7, 9, 19, 1]`, תת הסדרה החשבונית הרציפה הארוכה ביותר היא `[3,5,7,9]` ולכן יודפס שאורך הסדרה הוא 4, והפרש הסדרה הוא 2 (פורמט ההדפסה לבחירתכם).

```

/* @pre: arr.length >= 2 */
public void getLongestArrProg(int[] arr){
    // Default value is the sequence formed by elements 0 and 1
    int currDelta = arr[1] - arr[0];
    int currSeqLength = 2;
    int maxDelta = currDelta;
    int maxSeqLength = currSeqLength;

    for (int i = 2; i < arr.length; i++) {
        int delta = arr[i] - arr[i-1];

        if (currDelta == delta) { //continuing same sequence
            currSeqLength +=1;
        }
        else { //starting a new sequence
            currSeqLength=2;
            currDelta = delta;
        }
        //updating max sequence, if needed
        if (maxSeqLength < currSeqLength) {
            maxDelta = currDelta;
            maxSeqLength = currSeqLength;
        }
    }
    System.out.println(
        String.format("Sequence length=%d, delta=%d",
            maxSeqLength, maxDelta));
}
}

```

מסגרת חירום

שאלה 3 (5 נק':):

להלן מספר טענות, בחר'י בתשובה הטובה ביותר:

- א. ניתן לרשת ממחלקה המוגדרת כ record .
- ב. במחלקה אשר מוגדרת כ record לא ניתן לממש שירותי מופע.
- ג. במחלקה שהיא record לא ניתן להוסיף קוד לבנאי מעבר לקוד האטומטי שנוצר לאתחול השדות.
- ד. במחלקה שהיא record כל השדות מוגדרים כ final.
- ה. כל התשובות מלבד תשובה זו לא נכונות.
- ו. מלבד תשובה זו יש יותר מתשובה נכונה אחת.

נימוק:

בבדיקת הבחינה קיבלנו גם את תשובה ד' כנכונה. בניסוח של תשובה זו נפלה טעות, ונכתב "שדות" במקום "שדות מופע", שזה מה שהתכוונו. מכיוון שלא ראינו בקורס שדות סטטיים של records, לסטודנטים לא היתה כל דרך לדעת אם זה נכון שגם שדות סטטיים חייבים להיות מוגדרים כ final (הם לא), ולכן שתי התשובות התקבלו כנכונות.

שאלה 4 (5 נק':):

מה יודפס בהרצת התוכנית הבאה?

```
public class Q4 {
    public static int[] replaceArr(int[] arr) {
        arr = new int[] {4,5,6};
        return arr;
    }

    public static void replaceInArr(int[] arr) {
        arr[0] = 19;
    }

    public static void main(String[] args) {
        int[] arr1 = {1,2,3};
        int[] arr2 = {1,2,3};
        int[] arr3 = arr1;
        System.out.println(arr1 == arr2);
        replaceArr(arr2);
        System.out.println(arr2[0] == 4);
        replaceInArr(arr3);
        System.out.println(arr1[0] == 19);
    }
}
```

- 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 5 (5 נק')::

```
public class Grandma{
}

public class Mom extends Grandma{
    public Mom(int i1) {
        //super(i1);          // m1
    }
}

public class Child extends Mom{
    public Child(String str) {
        //this(1, 2);        // c1
        //super(str);        // c2
    }

    public Child(int i1, int i2) {
        super(i1+i2);
    }
}
```

בחר/י בתשובה הטובה ביותר:

- א. הקוד מתקמפל כמו שהוא.
- ב. הקוד לא מתקמפל. הוא יתקמפל אם נוציא את שורה m1 מההערה.
- ג. הקוד לא מתקמפל. הוא יתקמפל אם נוציא את שורה c1 מההערה.
- ד. הקוד לא מתקמפל. הוא יתקמפל אם נוציא את שורה c2 מההערה.
- ה. הקוד לא מתקמפל. הוא יתקמפל אם נוציא את שורות c1+m1 מההערה.
- ו. הקוד לא מתקמפל, הוא יתקמפל אם נוציא את שורות c2+m1 מההערה.
- ז. מלבד תשובה זו, כל התשובות לא נכונות.
- ח. מלבד תשובה זו יש יותר מתשובה נכונה אחת.

נימוק:

שאלה 6 (5 נק'):

```
public class Base {
    private boolean myBool = false;

    public void setMyBool(boolean b) { myBool = b; }
}

public class Sub extends Base { }
```

בחר/י בתשובה הטובה ביותר:

- א. תחת ההנחה ש Base היא מחלקה public וקיים עבודה לפחות בנאי public אחד, אם נרצה להגביל ירושה מ Base (למשל, לאסור על Sub לרשת ממנה), האופציה היחידה העומדת לרשותנו היא הגדרת Base כ final.
- ב. השדה myBool לא נורש ע"י Sub (כלומר, באובייקט שנוצר ע"י new Sub() לא ייוצר בזכרון השדה myBool).
- ג. אם נשנה את myBool להיות final, המחלקה Base תמשיך להתקמפל.
- ד. מלבד תשובה זו יש יותר מתשובה נכונה אחת.
- ה. מלבד תשובה זו כל התשובות לא נכונות.

נימוק:

שאלה 7 (5 נק'):

לפניכם 3 טענות המתייחסות למשתנים\שדות. שימו לב: כשאנחנו מתייחסים למקום שבו נשמרים משתנה\שדה, אנחנו מתייחסים למיקום המשתנה\שדה עצמו, ולא לערך המוצבע (אם מדובר במצביע).

- טענה 1: משתנה מקומי מטיפוס List נשמר על המחסנית (stack) ואינו מקבל ערך דיפולטי.
- טענה 2: שדה מופע מטיפוס List נשמר על הערימה (heap) ואינו מקבל ערך דיפולטי.
- טענה 3: שדה סטטי מטיפוס int מוקצה על המחסנית (stack) ומקבל ערך דיפולטי.

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 2+3 נכונות.
- ו. רק טענות 1+3 נכונות.
- ז. כל הטענות נכונות.
- ח. כל הטענות לא נכונות.

נימוק:

שאלה 8 (5 נק'): _____

```

public class Q8 {
    public static void main(String[] args) {
        test(new HashSet<>(100, 0.75));
        test(new TreeSet<>());
    }

    public static void test(Set<Box> mySet) {
        mySet.add(new Box("abc"));
        mySet.add(new Box("abc"));
        mySet.add(new Box("acd"));
        System.out.print(mySet.size() + " ");
    }
}

public class Box implements Comparable<Box>{
    private String str;
    private static int cnt = 1;

    public Box(String str) {this.str = str;}

    public int hashCode() { return cnt++; }

    public boolean equals(Object obj) {
        Box other = (Box) obj;
        return this.str.equals(other.str);
    }

    public int compareTo(Box o) {
        return Character.compare(this.str.charAt(0),
            o.str.charAt(0));
    }
}
    
```

השורה הראשונה מייצרת HashSet בגודל התחלתי של 100 (הפרמטר השני הוא פקטור הגדילה שלא רלוונטי לשאלה זו, וניתן להתעלם ממנו).

מה יודפס בהרצת הקוד הבא?

- א. 12
- ב. 13
- ג. 21
- ד. 22
- ה. 23
- ו. 31
- ז. 32
- ח. 33

נימוק:

שאלה 9 (5 נק'):

לפניכם המימוש של המחלקות Base ו Sub, וכן 3 פונקציות שנרצה לבחון את הוספתן למחלקה Base. נבחן הוספה של כל אחת מהפונקציות בנפרד.

```
public class Base{
    public Object func(Object str) { /* code here */;
}

public class Sub extends Base{
    //method candidate inserted here
}

//method candidates
public String func(String str) { /* code here */} //f1

private Object func(String str) { /* code here */} //f2

public Object func(Object str) throws Exception { /* code here */} //f3
```

אילו מהפונקציות ניתן להוסיף ל Sub כך שהקוד ימשיך להתקמפל? בחר/י בתשובה הטובה ביותר:

- א. רק אם נוסיף את f1 הקוד ימשיך להתקמפל.
- ב. רק אם נוסיף את f2 הקוד ימשיך להתקמפל.
- ג. רק אם נוסיף את f3 הקוד ימשיך להתקמפל.
- ד. רק אם נוסיף את f1 או את f2 הקוד ימשיך להתקמפל.
- ה. רק אם נוסיף את f2 או את f3 הקוד ימשיך להתקמפל.
- ו. רק אם נוסיף את f1 או את f3 הקוד ימשיך להתקמפל.
- ז. ניתן להוסיף כל אחת משלוש הפונקציות, והקוד עדין ימשיך להתקמפל.
- ח. לכל פונקציה שנוסיף, הקוד לא יתקמפל.

נימוק:

שאלה 10 (5 נק'):

נרצה להגדיר את המחלקה MyMap כך שהקוד הבא יתקמפל:

```
public static void main(String[] args) {
    MyMap<String> strings = new MyMap<>();
    MyMap<Integer> ints = new MyMap<>();
    strings.put("abc", "de");
    ints.put(1, 2);
    Integer i = ints.get(1);
}
```

לפניכם שלוש אופציות להגדרת MyMap:

אופציה 1: `public class MyMap<T, V> extends HashMap<T, V> {}`

אופציה 2: `public class MyMap<T> extends HashMap<T, T> {}`

אופציה 3: `public class MyMap<T> extends HashMap<T, Object> {}`

נרצה לבחור באופציה שמתקמפלת בעצמה, וגם שהקוד של הפונקציה main יתקמפל..

- א. רק אופציה 1.
- ב. רק אופציה 2.
- ג. רק אופציה 3.
- ד. רק אופציה 1 או אופציה 2.
- ה. רק אופציה 1 או אופציה 3.
- ו. רק אופציה 2 או אופציה 3.
- ז. לכל אחת מהאופציות, הקוד יתקמפל.
- ח. לכל אחת מהאופציות, הקוד לא יתקמפל.

נימוק:
אופציה 3 לא מתקמפלת כמו שהיא, אבל גם התשובה ה' קיבלה ניקוד מלא בהנתן נימוק שהתעלם מהמימוש הריק והציע מימוש שבו MyMap דורסת את get ומשנה את ערך ההחזרה.

שאלה 11 (5 נק'): _____

לפניכם 3 טענות הקשורות לטיפוסים גנריים:

טענה 1: לאחר מחיקת הטיפוסים (erasure), בכל מחלקה גנרית שבה מוגדר טיפוס גנרי T כלשהו, הטיפוס הסטטי של כל שדה\משתנה שהיה מוגדר כ T הופך ל Object.

טענה 2: ניתן להחליף את הכוכביות בקוד המצורף בבלוק קוד שמתקמפל ורץ ומצליח להכניס מחרוזת ל lst.

```
List<Integer> lst = new ArrayList<>();  
/*****/
```

טענה 3: הקוד של המחלקה Inner מתקמפל.

```
class Gen<T>{  
    public static class Inner{  
        public T t;  
    }  
}
```

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.

נימוק:

שאלה 12 (5 נק')
שאלה 12 (5 נק')

```
public class Base{
    public int sum = 0;
    public static int i = 2;

    public Base(){
        sum = i + 2*getNum();
    }
    public int getNum() { return i; }
}

public class Sub extends Base{
    public static int i = 1;

    public Sub() {
        sum = sum + i + this.getNum();
    }

    public int getNum() { return i; }

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

מה יודפס בהרצתה תוכנית Sub?

- א. 5
- ב. 6
- ג. 7
- ד. 8
- ה. 9
- ו. 10
- ז. 11
- ח. 12

נימוק:

`public interface Map<K,V>`

Modifier and Type	Method and Description
boolean	<code>containsKey(Object key)</code> Returns true if this map contains a mapping for the specified key.
<code>Set<Map.Entry<K,V>></code>	<code>entrySet()</code> Returns a <u>Set</u> view of the mappings contained in this map.
V	<code>get(Object key)</code> Returns the value to which the specified key is mapped, or <u>null</u> if this map contains no mapping for the key.
V	<code>getOrDefault(Object key, V defaultValue)</code> Returns the value to which the specified key is mapped, or <u>defaultValue</u> if this map contains no mapping for the key.
boolean	<code>isEmpty()</code> Returns true if this map contains no key-value mappings.
<code>Set<K></code>	<code>keySet()</code> Returns a <u>Set</u> view of the keys contained in this map.
V	<code>put(K key, V value)</code> 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	<code>remove(Object key)</code> 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	<code>size()</code> Returns the number of key-value mappings in this map.
<code>Collection<V></code>	<code>values()</code> Returns a <u>Collection</u> view of the values contained in this map.

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

boolean	<code>add(E e)</code> Adds the specified element to this set if it is not already present.
boolean	<code>addAll(Collection<? Extends E> c)</code> 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.
boolean	<code>contains(Object o)</code> Returns true if this set contains the specified element.
boolean	<code>isEmpty()</code> Returns true if this set contains no elements.
<code>Iterator<E></code>	<code>iterator()</code> Returns an iterator over the elements in this set.
boolean	<code>remove(Object o)</code> Removes the specified element from this set if it is present.
int	<code>size()</code> Returns the number of elements in this set (its cardinality).

עמוד 20 מתוך 20

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	add(int index, E e) Inserts the specified element at the specified position in 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
E	remove(int index) Removes the element at the specified position in this list. Returns the element previously at the specified position
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 <u>Comparator</u> .

public final class String

char	charAt(int index) Returns the <u>char</u> value at the specified index.
static String	join(CharSequence delimiter, CharSequence... elements) Returns a new String composed of copies of the <u>CharSequence elements</u> joined together with a copy of the specified <u>delimiter</u> .
int	length() Returns the length of this string.
String[]	split(String regex) Splits this string around matches of the given regular expression.
String	substring(int beginIndex) Returns a string that is a substring of this string.
char[]	toCharArray() Converts this string to a new character array.

שימוש ב records: עבור record שהוגדר באופן הבא:

```
public record Location(int row, int col) {}
```

מצורת דוגמת קוד שמייצרת Location וניגשת לשני השדות שלו:

```
Location loc = new Location(3, 5);  
int x = loc.row();  
int col = loc.col();
```