

# פתרון הבחינה בתוכנה 1

## סמסטר ב', מועד ב' 2013/2014

### 13.8.2014

דן הלפרין, שחר מעוז, ליאור וולף, יעל אמסטרדמר, דביר נתנאלי ולנה דנקין

### הוראות (נא לקרוא!)

- משך הבחינה **שלוש שעות**, חלקו את זמנכם ביעילות.
- אסור השימוש בחומר עזר כלשהו, כולל מחשבוניו או כל מכשיר אחר פרט לעט. בסוף הבחינה צורף לנוחותכם נספח ובו תיעוד מחלקות שימושיות.
- יש לענות על כל השאלות בגוף הבחינה במקום המיועד לכך. המקום המיועד מספיק לתשובות מלאות. יש לצרף את טופס המבחן למחברת הבחינה. מחברת ללא טופס המבחן תיפסל. תשובות במחברת הבחינה לא תיבדקנה. במידת הצורך ניתן לכתוב בגב טופס הבחינה.
- יש למלא מספר סידורי (מס' מחברת) ומספר ת.ז על כל דף של טופס הבחינה.
- ניתן להניח לאורך השאלה שכל החבילות הדרושות יובאו, ואין צורך לכתוב שורות import.
- במקומות בהם תתבקשו לכתוב מתודה (שירות), ניתן לכתוב גם מתודות עזר.
- ניתן להוסיף הנחות לגבי אופן השימוש בשירותים המופיעים בבחינה, ובלבד שאין הן סותרות את תנאי השאלה. יש לתעד הנחות אלו כחוזה (תנאי קדם, תנאי בתר) בתחביר המקובל, שייכתב בתחילת השירות.
- יש להתייחס בכבוד רב לצוות המשיחים.

לשימוש הבודקים בלבד:

שאלה	משקל	א	ב	ג	ד	ה	סה"כ
1	40						
2	24						
3	20						
4	16						

## בהצלחה!

כל הזכויות שמורות ©  
 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.

**שאלה 1 (40 נקודות)**

בשאלה זו עליכם לממש חלק מספרייה לניתוח של לוג אירועים עבור ריצה של מערכת. הספרייה מאפשרת לקרוא לוג אירועים בעל פורמט קבוע מקובץ טקסט נתון ולאחר מכן לבצע ניתוחים שונים על המידע. הפורמט של כל שורה בלוג האירועים הוא:

TIMESTAMP EVENT\_NAME

כאשר TIMESTAMP הוא מספר חיובי שלם, לאחריו מופיע רווח בודד, ולאחר מכן, EVENT\_NAME שהוא שם של אירוע המיוצג ע"י רצף של תווים ללא רווחים.

לדוגמא, השורות בקובץ יכולות להיות

```
3 start
8 close
4 open
```

**סעיף א' (15 נק')**

השלימו את מימוש המתודה הסטטית readLog במחלקה LogAnalyzer.

המתודה מקבלת מחרוזת המייצגת מסלול לקובץ לוג (אשר שמור בפורמט שהוגדר למעלה). היא מאחסנת את שמות האירועים מהקובץ בתוך רשימת מחרוזות המוחזקת כשדה סטטי בשם events במחלקה.

על הרשימה להיות ממוינת על פי הערכים של שדה ה-TIMESTAMP בסדר עולה. שימו לב, לא מובטח שזהו בהכרח הסדר בו האירועים הופיעו בקובץ - למשל, בדוגמא למעלה, 8 מופיע לפני 4.

**טיפול בקלט לא תקין:** בכל מקרה של בעיה בקריאת הלוג, המתודה readLog תחזיר false, והשדה events יהיה זהה לערך שהיה לו לפני הקריאה למתודה (כלומר, הוא לא ישתנה). מקרים אלה כוללים:

- TIMESTAMP שחוזר פעמיים או יותר בקובץ
- שורה שאינה מכילה בדיוק שני שדות (אבל ניתן להניח שאם כן, השדה הראשון ניתן להמרה ל-int)
- חריג שנזרק במהלך העבודה עם הקובץ

```
public class LogAnalyzer {
    private static List<String> events = null;

    public static boolean readLog(String inputFilePath) {
        Map<Integer, String> timeStampToRowMap = new TreeMap<Integer, String>();
        File file = new File(inputFilePath);
        try (Scanner sc = new Scanner(file)) {
            while (sc.hasNextLine()) {
                String line = sc.nextLine();
                String[] linesStrings = line.split(" ");
                if (linesStrings.length != 2) { // check num of words in a row
                    return false;
                }
                int timestamp = Integer.parseInt(linesStrings[0]);

                if (timeStampToRowMap.containsKey(timestamp)) {
                    return false; //duplicate timestamp
                } else {
                    timeStampToRowMap.put(timestamp, line);
                }
            }
            if (sc.ioException() != null) {
                //error while reading from scanner
                return false;
            }
        } catch (FileNotFoundException e) {
            //error while reading from scanner
            return false;
        }
        events = new ArrayList<String>();
        for (Integer key : timeStampToRowMap.keySet()) {
            events.add(timeStampToRowMap.get(key).split(" ")[1]);
        }
        return true;
    }
}
```

}

סעיף ב' (15 נק')

הגדרה: תבנית אירועים היא רשימת שמות אירועים לא ריקה וללא חזרות על אותו אירוע. **מופע של תבנית אירועים** בתוך רשימה X הוא תת-רשימה של X המקיימת את התנאים הבאים:

1. האירועים מהתבנית מופיעים בתת-הרשימה על פי סדרם בתבנית.
2. תת-הרשימה מתחילה באירוע הראשון בתבנית ומסתיימת באירוע האחרון בתבנית.
3. בנוסף, בין האירועים מהתבנית, יכולים להופיע בתת-הרשימה אירועים נוספים, אך רק בתנאי שהם אינם מופיעים בתבנית כלל.

לדוגמא, לתבנית האירועים `<open,read,close>` יש שני מופעים ברשימה  
`<open,open,read,end,close,end,open,start,read,start,end,close,open,read,read,close>`  
 (המופעים מסומנים בדוגמא זו בקו תחתון).

בסעיף זה נממש את המחלקה `LogPattern` אשר מייצגת תבנית אירועים. ניתן להזין לתוך מחלקה זאת בכל פעם את האירוע הבא ברשימה, והמחלקה תעקוב אחרי רצפי אירועים ותזהה מופעים של התבנית ברשימה.

- הגדירו את שדות המחלקה.

- ממשו בנאי אשר מקבל כקלט תבנית אירועים כרשימה (אשר ניתן להניח שהיא תקינה -- לא ריקה וללא חזרות).

- ממשו מתודה בשם `visitEvent`. מתודה זו מקבלת כקלט את שם האירוע הבא ברשימה, ומחזירה `true` אם"ם אירוע זה הוא **האחרון במופע של התבנית** שאיתה אותחל בנאי המחלקה (כלומר, האחרון בתת-הרשימה). על כן, הערך שמחזירה המתודה `visitEvent` תלוי לא רק בשם האירוע שהיא קיבלה כארגומנט אלא גם בקריאות הקודמות למתודה.

לדוגמא, נעביר לבנאי של `LogPattern` את התבנית `<open,read,close>`, ולאחר מכן נקרא למתודה `visitEvent` 16 פעמים, בכל פעם עם אירוע אחר מתוך הרשימה מהדוגמא למעלה, על פי סדר האירועים החל מ-`open` הראשון וכלה ב-`close` האחרון. הקריאה ה-5 (עם `close`) וה-12 (עם `close`) יחזירו `true` מכיוון שהן מייצגות אירועים אחרונים במופע של התבנית, וכל יתר הקריאות יחזירו `false`.

✓ רמז: כדי לממש את `visitEvent` על המחלקה `LogPattern` להחזיק לפחות שני שדות פרטיים. האחד לשמירת התבנית שניתנה לבנאי. השני לשמירת המיקום הנוכחי של הקריאות ל `visitEvent` על גבי התבנית.

```
public class LogPattern {
    // write here fields and constructor
    protected List<String> events = new ArrayList<>();
    private Set<String> eventNames = new HashSet<>();
    private int currPos;

    public LogPattern(List<String> eventsList) {
        events.addAll(eventsList);
        eventNames.addAll(eventsList);
        currPos = 0;
    }

    public boolean visitEvent(String eventName) {
        if (events.get(currPos).equals(eventName)) {
            //the current event fits the current pattern instance
            currPos++;
            if (currPos == events.size()) {
                // the event was the last one in the pattern
                currPos = 0;
                return true;
            }
        } else if (eventNames.contains(eventName)) {
            //the current event does not fit the current pattern instance
            if (events.get(0).equals(eventName)) {
                //the current event may start a new pattern instance
                currPos = 1;
            } else {
                currPos = 0;
            }
        }
        return false;
    }
}
```

## סעיף ג' (10 נק')

ממשו מחלקה חדשה בשם `LogPatternWithCounter` היורשת מהמחלקה `LogPattern` שמימשתם בסעיף הקודם. למחלקה החדשה שתי מתודות נוספות על אלו של `LogPattern`. האחת, בשם `getEvents`, מחזירה את תבנית האירועים שהתקבלה בבנאי. השנייה, בשם `getEventsCounter`, מחזירה את מספר הפעמים שהמתודה `visitEvent` החזירה ערך `true`.

✓ רמז: כיצד נספור כמה פעמים `visitEvent` החזירה `true`? לשם כך, כדאי לדרוס את המתודה `visitEvent`, ולהוסיף מנגנון של קידום ה-`counter`. כיצד נמנע משכפול קוד של `visitEvent` שהוגדר במחלקת האב? לשם כך ניתן לקרוא מתוך המתודה הדורסת למימוש שהוגדר במחלקת האב.

```
public class LogPatternWithCounter extends LogPattern {
    // fields, constructor - if necessary
```

```
    int cnt = 0;
```

```
    public LogPatternWithCounter(List<String> eventsList) {
        super(eventsList);
    }
}
```

```
    public List<String> getEvents() {
```

```
        return events;
    }
}
```

```
    public int getEventsCounter() {
```

```
        return cnt;
    }
}
```

```
    public boolean visitEvent(String eventName) {
```

```
        if (super.visitEvent(eventName)) {
```

```
            cnt++;
```

```
            return true;
        }
    }
}
```

```
        return false;
    }
}
```

```
    }
}
```

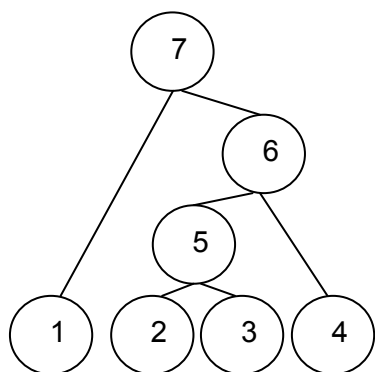
## שאלה 2 (24 נקודות)

בשאלה זו נייצג עצים בינאריים באמצעות מטריצה דו מימדית. עץ בינארי עם  $m$  עלים ייוצג באמצעות מטריצה עם  $(m-1)$  שורות ושתי עמודות. מטריצה זו נקראת ZTREE.

לעלים של עץ זה ניתן מספרים מזהים (IDs) בין 1 ל- $m$ . שתי העמודות בשורה מסוימת של המטריצה מכילות מזהים של צמתים שמחוברים בשורה הזאת כדי ליצור תת עץ משותף (סדר המזהים בשורה אינו משנה). השורה ה- $i$  של המטריצה מייצגת את שני הילדים של הצומת בעל מזהה  $m+i+1$ . צומת מופיע בטבלה רק אחרי שילדיו הופיעו (אלא אם הם עלים), ולכן המזהה שלו בהכרח גבוה יותר.

למשל, נניח שיש 4 עלים. המזהים שלהם יהיו המספרים מ 1-4. אם בשורה הראשונה (שורה באינדקס 0) של ZTREE מופיעים האינדקסים 3 ו-2, נוצר תת עץ שמחבר את עלים 3 ו-2 עם המזהה  $4+0+1 = 5$ .

עבור העץ המצויר כאן מתאימה המטריצה:



←	2	3	(שורה 0 מייצגת את תתי העצים של הצומת עם המזהה 5)
←	5	4	(שורה 1 מייצגת את תתי העצים של הצומת עם המזהה 6)
←	1	6	(שורה 2 מייצגת את תתי העצים של הצומת עם המזהה 7)

### סעיף א' (10 נק')

ממשו איטרטור אשר מקבל בבנאי שלו מטריצה דו ממדית המייצגת ZTREE. המטריצה נתונה בתור מערך דו ממדי של מספרים שלמים. למשל, בדוגמא למעלה, המטריצה יכולה להיות:

```
int [][] exampleZTREE = {{2,3}, {5,4}, {1,6}};
```

האיטרטור מחזיר את הצמתים, החל מהעלים ועד ראש העץ, בסדר שבו כל צומת מוחזר רק אחרי ששני ילדיו הוחזרו. הפעם, נייצג כל צומת ע"י מערך של שלושה int-ים: המזהה של תת הצומת, והמזהים של שני ילדיו (אין חשיבות לסדר הילדים בשלשה). מכיוון שלעלים אין ילדים, נחזיר 0 בתור מזהי הילדים של עלים.

להלן דוגמת הרצה של האיטרטור על המטריצה שהגדרנו למעלה:

```

Iterator<int[]> iter = new ZTreeIterator(exampleZTREE);
int[] triple = iter.next(); // returns {1, 0, 0}
triple = iter.next();     // returns {2, 0, 0}
triple = iter.next();     // returns {3, 0, 0}
triple = iter.next();     // returns {4, 0, 0}
triple = iter.next();     // returns {5, 3, 2}
triple = iter.next();     // returns {6, 5, 4}
triple = iter.next();     // returns {7, 1, 6}
  
```

יתכנו ריצות אחרות שגם הן חוקיות, למשל ע"י שינוי סדר הופעת העלים בפלט.

```
class ZTreeIterator implements Iterator<int[]> {

    int[][] matrix;
    int currNode = 1;
    int numOfLeaves;

    public ZTreeIterator(int[][] matrix) {
        this.matrix = matrix;
        this.numOfLeaves = matrix.length + 1;
    }

    @Override
    public boolean hasNext() {
        return currNode < ((numOfLeaves) * 2);
    }

    @Override
    public int[] next() {
        if (currNode <= numOfLeaves) { // this is a leaf
            return new int[] { currNode++, 0, 0 };
        } else {
            //the first row in the table (with index 0)
            //represents the sub tree with id: 1 + numOfLeaves
            int[] sons = matrix[currNode - numOfLeaves - 1];
            return new int[] { currNode++, sons[0], sons[1] };
        }
    }

    @Override
    public void remove() {
        // Empty implementation
    }
}
```



סעיף ב' (7 נק')

המתודה הסטטית `countLeaves` מקבלת ZTREE בתור מערך דו ממדי של `int`-ים ומזהה של צומת ומחזירה את מספר העלים בתת העץ של הצומת. אם הצומת הוא עלה, מספר העלים בתת העץ שלו הוא 1. לדוגמא:

```
countLeaves(exampleZTREE,2); // returns 1
countLeaves(exampleZTREE,6); // returns 3
countLeaves(exampleZTREE,7); // returns 4
```

השלימו את מימוש המתודה. ניתן להוסיף מתודות עזר לפי הצורך במלבן למטה (מומלץ לקרוא תחילה גם את הסעיף הבא).

```
public static int countLeaves(int[][] matrix, int nodeId) {
```

```
    if (isLeaf(matrix, nodeId)) {
        return 1;
    }
    int[] sons = getSonsForNodeID(matrix, nodeId);
    return countLeaves(matrix, sons[0])
        + countLeaves(matrix, sons[1]);
}
```

```
//auxiliary methods
private static boolean isLeaf(int[][] matrix, int nodeId) {
    return nodeId <= getNumOfLeaves(matrix);
}

private static int getNumOfLeaves(int[][] matrix) {
    return matrix.length + 1;
}

private static int[] getSonsForNodeID(int[][] matrix, int nodeId){
    return matrix[nodeID-getNumOfLeaves(matrix)-1];
}
```

סעיף ג' (7 נק')

גובה של צומת הוא אורך המסלול המקסימלי מצומת לעלה בתוך תת העץ. כתבו מתודה סטטית אשר מקבלת ZTREE בתור מערך דו ממדי של int-ים ומזהה של צומת ומחזירה את גובה הצומת הנתון. גובה של עלה הוא אפס.

```
getHeight(exampleZTREE,2); // returns 0  
getHeight(exampleZTREE,6); // returns 2  
getHeight(exampleZTREE,7); // returns 3
```

```
public static int getHeight(int[][] matrix, int nodeId) {
```

```
    if (isLeaf(matrix, nodeId)) {  
        return 0;  
    }  
    int[] sons = getSonsForNodeID(matrix, nodeId);  
    return 1 + Math.max(getHeight(matrix, sons[0]),  
                       getHeight(matrix, sons[1]));
```

```
}
```

**שאלה 3 (20 נקודות)**

התבוננו בקוד המחלקות הבאות וענו על השאלות.

```
public class SubException extends Exception {
    public void printStackTrace(PrintStream s) {
        System.out.println("sub!");
    }
}
```

```
public class SubSubException extends SubException {
    public List<Object> lst = new LinkedList<>();
    public void printStackTrace() {
        System.out.println("subsub!");
    }
}
```

```
public class Main {
    public static void main(String[] args) throws Exception {
        SubSubException subsub = new SubSubException();
        try {
            throw subsub;
        }
        /* */
        System.out.println("bye");
    }
}
```

בכל אחד מהסעיפים הבאים מוחלף סימון ה- /\* \*/ בפונקציית ה-main במס' שורות קוד. הנכם מתבקשים לציין מהו הפלט של הרצת פונקציית ה-main בכל אחד מהמקרים. אם לדעתכם התכנית אינה עוברת קומפילציה או זורקת שגיאה בזמן הסבירו ממה השגיאה נובעת.

**שימו לב:**

- בכל מקרה (שגיאה או ריצה תקינה) יש לכתוב הסבר קצר.
- תזכורת: כאשר קוראים ל- throw של חריג מטיפוס XXX מתוך try, ניתן לתפוס אותו ולטפל בו או להניח לו להיזרק הלאה, באותו אופן כמו קריאה למתודה מתוך try אשר מצהירה על זריקת חריג מטיפוס XXX. ההבדל הוא שבקריאה למתודה לא מובטח שייזרק חריג.
- נתון כי כאשר נזרק חריג החוצה מפונקציית ה-main (ולא מטופל), נקראת המתודה שלו printStackTrace(PrintStream s) אשר מדפיסה ל-Stream הנתון (למשל System.out, שהוא הזרם המדפיס לפלט הסטנדרטי של התכנית).
- במקרה שאינכם יודעים מה מדפיס חריג מסוים (שעבורו הקוד אינו נתון) ניתן לציין בנק' המתאימה "printStackTrace של חריג מסוג XXX" (נניח, NullPointerException).
- תיעוד מלא של המחלקה Exception מצורף בנספח.
- נתון כי המחלקות SubException ו-SubSubException מתקמפלות ללא שגיאות.

סעיף א' (4 נק')  
- /\* \*/ מוחלף ב-

```
catch (RuntimeException e) {
    System.out.println("runtime!");
}
```

**תקבל שגיאה בזמן ריצה (יודפס: sub!)**

SubSubException אינו יורש מ-RuntimeException. לכן הוא לא ייתפס אלא יזרק החוצה מה-main. לכן גם לא יודפס .bye

סעיף ב' (4 נק')  
- /\* \*/ מוחלף ב-

```
catch (SubException e) {
    e.printStackTrace(System.out);
} catch (Exception e) {
    System.out.println("exception!");
}
```

**התכנית תרוץ ללא שגיאה ותדפיס: sub! bye**

החריג ייתפס בבולק catch הראשון שמתאים לו - כלומר, בבולק הראשון. מה שיודפס הוא לפי המימוש ב-SubException של printStackTrace(PrintStream) (המתודה ב-SubSubException היא העמסה ולא דריסה). לבסוף יודפס .bye

סעיף ג' (4 נק')  
- /\* \*/ מוחלף ב-

```
catch (Exception e) {
    e.printStackTrace(System.out);
} finally {
    System.out.println("finally");
}
```

**התכנית תרוץ ללא שגיאה ותדפיס: sub! finally bye**

החריג ייתפס בבולק ה-catch היחיד ויטופל שם. המתודה ב-SubException דורסת את זאת של Exception ולכן printStackTrace ידפיס את השורה הראשונה. אח"כ ייקרא בולק ה-finally ובסוף יודפס .bye

סעיף ד' (4 נק')  
 /\* \*/ מוחלף ב-

```
catch (SubSubException e) {
    SubSubException e2 = new SubSubException();
    if (e2.lst == subsub.lst) {
        System.out.println("if");
    } else {
        System.out.println("else");
    }
}
```

**התכנית תרוץ ללא שגיאה ותדפיס:**  
 else  
 bye

האופרטור == מחזיר true רק על שני מצביעים של אותו עצם בזיכרון. מכיוון ש-lst ב-SubSubException הוא שדה מופע, לכל מופע יש שדה אחר. לכן לאחר שהחריג ייתפס ב-catch, ייקרא בלוק ה-else. לבסוף יודפס .bye.

סעיף ה' (4 נק')  
 /\* \*/ מוחלף ב-

```
catch (Exception e) {
    List<Object> lst = subsub.lst;
    subsub = new SubException();
    System.out.println(lst.size());
}
```

**תקבל שגיאת קומפילציה בשורה השלישית:** Type mismatch  
 לא ניתן לבצע השמה של SubException ל-SubSubException.

### שאלה 4 (16 נקודות)

בשאלה זו נעסוק בתוכנית המייצרת ממשק משתמש גרפי הכולל חלון ובו 3 כפתורי לחיצה (ראו תמונה למטה). במקטע הקוד המובא בהמשך, המתודה `createShell` מכילה מקטע קוד חסר. בכל אחד מהסעיפים הבאים, יחליף מקטע קוד מסוים את מקטע הקוד החסר.

```
public class GUIMain {
    private static Display display;
    private static int num = 0;

    public static void main(String[] args) {
        Shell shell = createShell();
        eventLoop(shell);
    }

    private static void eventLoop(Shell shell) {
        shell.pack();
        shell.open();
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                display.sleep();
            }
        }
        display.dispose();
    }

    private static Shell createShell() {
        display = Display.getDefault();
        Shell shell = new Shell(display);
        shell.setLayout(new RowLayout(SWT.HORIZONTAL));

Missing code



        return shell;
    }
}
```



#### הערות:

- ניתן להניח שהקוד (לאחר השלמת המקטע החסר) עובר קומפילציה ורץ ללא זריקת חריג.
- ה- `layout` בקוד `RowLayout(SWT.HORIZONTAL)` יגרום ליצירת הכפתורים משמאל לימין.
- **רצף לחיצה** מתאר מצב שבו לאחר פתיחת החלון של הממשק הגרפי, המשתמש לוחץ על הכפתורים בסדר מסוים. למשל, רצף הלחיצה  $3 \rightarrow 2 \rightarrow 1$  מתאר: לחיצה בודדת על הכפתור הראשון (השמאלי ביותר), לאחר מכן על הכפתור השני (האמצעי) ולבסוף לחיצה על הכפתור השלישי (הימני ביותר).

סעיף א' (4 נק')

אם נציב במקטע הקוד החסר במתודה createShell את מקטע הקוד הבא:

```


for (int i = 0; i < 3; i++) {
    Button b = new Button(shell, SWT.PUSH);
    b.setText(" ? ");
    b.addSelectionListener(new SelectionAdapter() {
        boolean clicked = false;

        public void widgetSelected(SelectionEvent e) {
            if (!clicked) {
                Button b = (Button) e.getSource();
                b.setText("Clicked!");
                clicked = true;
            }
        }
    });
}

```

ציינו בכתב על כל כפתור, מה יהיה הכיתוב שיופיע עליו לאחר רצפי הלחיצה הבאים (הפתרון של סעיף i כבר נתון לכם, בתור דוגמא):

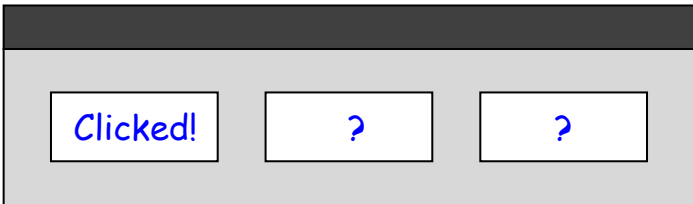
**דוגמא**



i. 1 → 2 → 3



ii. 3 → 2 → 2



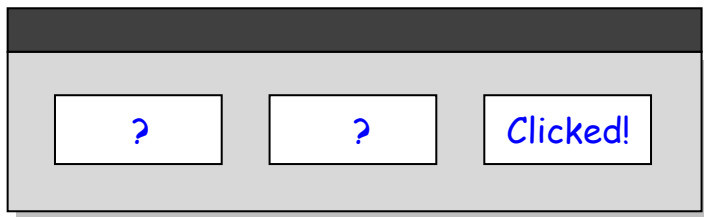
iii. 1 → 1 → 1

סעיף ב' (6 נק')

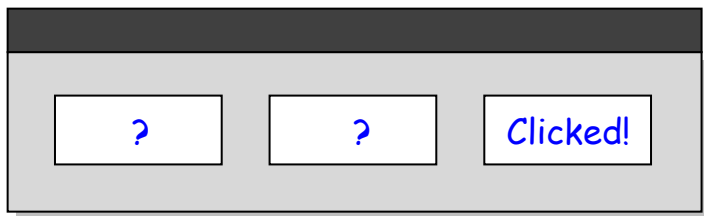
אם נציב במקטע הקוד החסר במתודה createShell את מקטע הקוד הבא:

```
final Button[] barr = new Button[3];
barr[0] = new Button(shell, SWT.PUSH);
barr[0].setText(" ? ");
for (int i = 1; i < 3; i++) {
    SelectionAdapter a = new SelectionAdapter() {
        boolean clicked = false;
        public void widgetSelected(SelectionEvent e) {
            if (!clicked) {
                Button b = (Button) barr[barr.length - 1];
                b.setText("Clicked!");
                clicked = true;
            }
        }
    };
    barr[i] = new Button(shell, SWT.PUSH);
    barr[i].setText(" ? ");
    barr[i].addSelectionListener(a);
}
```

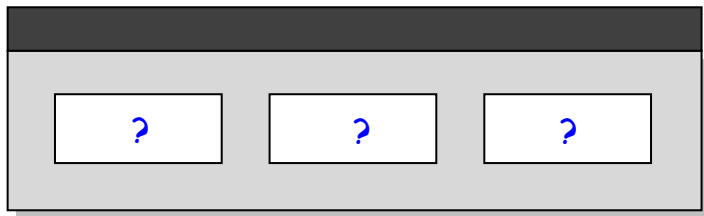
ציינו בכתב על כל כפתור, מה יהיה הכיתוב שיופיע עליו לאחר רצפי הלחיצה הבאים:



i. 1 → 2 → 3



ii. 3 → 2 → 2



iii. 1 → 1 → 1



סעיף ג' (6 נק')

אם נציב במקטע הקוד החסר במתודה createShell את מקטע הקוד הבא:

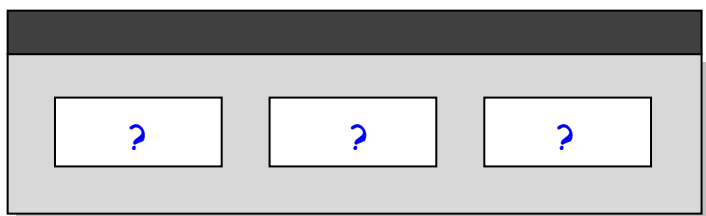
```
final Button[] barr = new Button[4];
for (int i = 0; i < 3; i++) {
    barr[i] = new Button(shell, SWT.PUSH);
    barr[i].setText(" ? ");
    barr[i].addSelectionListener(new SelectionAdapter() {
        boolean clicked = false;

        public void widgetSelected(SelectionEvent e) {
            Button b = (Button) e.getSource();
            if (barr[num] == b) {
                b.setText("Clicked!");
                num++;
            } else if (b.getText().equals("Clicked!")) {
                b.setText(" ? ");
            }
        }
    });
}
```

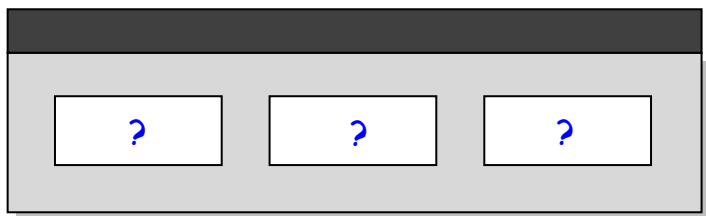
ציינו בכתב על כל כפתור, מה יהיה הכיתוב שיופיע עליו לאחר רצפי הלחיצה הבאים:



i. 1 → 2 → 3



ii. 3 → 2 → 2



iii. 1 → 1 → 1