פתרון בחינה בתוכנה 1

סמסטר ב', מועד א' ,תשע"א
26/6/2011

ליאור וולף, מתי שמרת, אסף זריצקי, הדס צור

הוראות (נא לקרוא!)

- משך הבחינה **שלוש שעות**, חלקו את זמנכם ביעילות.
- אסור השימוש בחומר עזר כלשהו, כולל מחשבונים או כל מכשיר אחר פרט לעט.
- בסוף הבחינה צורף לנוחותכם נספח ובו תיעוד מחלקות שימושיות.
- יש לענות על כל השאלות בטופס הבחינה במקום המיועד לכך. המקום המיועד מספיק לתשובות מלאות. יש לצרף את טופס המבחן למחברת הבחינה. מחברת ללא טופס עזר 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 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 השרות.

לשימוש הבודקים בלבד:

סעיף / שאלה	א	ב	ג	ד	
1					
2					
3					
סה"כ					

בהצלחה!

שאלה 1 (60 נקודות)

א. **(15 נקודות)** Logger משמש לשמירת הודעות בזמן הרצת תכנית מחשב. ניתן לשמור את ההודעות בקובץ, בזיכרון, במסד נתונים ועוד. לפניכם המנשק ILogger המאפשר שתי פעולות: (1) רישום הודעה ו-(2) הדפסת n ההודעות האחרונות שנשמרו.

```
public interface ILogger {
    /**
     * Adds the new message to the end of the log.
     */
    public void logThis(String message);

    /**
     * Prints the last n messages from the log, or less if there
     * are not enough messages in the logger.
     */
    public void printLast(int n);
}
```

ממשו את המחלקה SimpleLogger המממשת את המנשק הנ"ל באמצעות מערך של הודעות. גודל המערך (מספר ההודעות המירבי) קבוע וניתן בבנאי.כאשר המערך מתמלא "שוכחים" את ההודעה הישנה ביותר.

שימו לב: על מנת שהמימוש יהיה יעיל **אין להזיז הודעות במערך** כאשר מכניסים הודעה חדשה.

רמז: ניתן להשתמש במיקום במערך של הודעה קיימת.

```
public class SimpleLogger ______________________________ {
```

מקום נוסף בעמוד הבא

```java
public class SimpleLogger implements ILogger {
    private String[] log;
    private int next;

    public SimpleLogger(int capacity) {
        log = new String[capacity];
        next = 0;
    }

    @Override
    public void logThis(String str) {
        if (str == null) {
            throw new IllegalArgumentException("str == null");
        }
        log[next++ % log.length] = str;
    }

    @Override
    public void printLast(int n) {
        if (n < 0 || n > log.length) {
            throw new IllegalArgumentException("illegal n");
        }
        n = Math.min(n, next);
        int index = n - 1;
        for (int i = 0; i < n; i++) {
            if (index < 0) {
                index = log.length - 1;
            }
            System.out.println(log[index--]);
        }
    }
}
```

ב. **(25 נקודות)** המחלקה ThreadLocal משמשת ליצירת מבנה נתונים המכיל ומתחזק עותק של משתנה מטיפוס מסוים לכל חוט (thread). כלומר שימוש במחלקה מאפשר להגדיר משתנה פעם אחת ועם זאת להבטיח שלכל חוט יהיה את העותק שלו וכן שחוט לא יוכל לגשת לעותק שאינו שלו. אובייקט מטיפוס ThreadLocal, מכיל את העותקים עבור כל החוטים, ודואג לכך שכל חוט יתייחס אך ורק לעותק הייחודי שמותאם אליו. כך כל חוט יכול לשנות את המשתנה מבלי לפגוע בערך המשתנה עבור החוטים האחרים. המחלקה משתמשת במנגנון הגנריות כדי לאפשר ליצור משתנים מכל סוג.

לפניכם התיעוד של המחלקה ThreadLocal:

Constructor Summary

ThreadLocal()   Creates a thread local variable.

Method Summary

T	**get**() Returns the value in the current thread's copy of this thread-local variable. If the variable has no value for the current thread, it is first initialized to the value returned by an invocation of the initialValue() method.
protected T	**initialValue**() Returns the current thread's "initial value" for this thread-local variable. This method will be invoked the first time a thread accesses the variable with the get() method, unless the thread previously invoked the set(T) method, in which case the initialValue method will not be invoked for the thread. Normally, this method is invoked at most once per thread, but it may be invoked again in case of subsequent invocations of remove() followed by get(). This implementation simply returns *null*; if the programmer desires thread-local variables to have an initial value other than null, ThreadLocal must be subclassed, and this method overridden.
void	**remove**() Removes the current thread's value for this thread-local variable. If this thread-local variable is subsequently read by the current thread, its value will be reinitialized by invoking its initialValue() method, unless its value is set by the current thread in the interim.
void	**set**(T value) Sets the current thread's copy of this thread-local variable to the specified value.

דוגמת שימוש: נרצה להגדיר משתנה המשותף לאובייקטים באותו חוט, אך לא לאובייקטים בחוטים אחרים. לשם כך נגדיר משתנה סטטי מטיפוס ThreadLocal באופן הבא:

```
private static ThreadLocal<Integer> myStaticThreadLocalInteger =
        new ThreadLocal<Integer>();
```

כעת, כל חוט יכול לפנות אל העותק "שלו", לקרוא את ערכו (get) ולשנות אותו (set), מבלי לשנות את העותקים של החוטים האחרים.
אם נרצה לממש מזהה מספרי (מטיפוס Integer) השונה מחוט לחוט, אזי נוכל להשתמש בהגדרה הנ"ל

```
public class ThreadLocalExample {

    private static final ThreadLocal<Integer> uniqueNum =
      new ThreadLocal<Integer>();

    public static int getIdentifier() {
       return uniqueNum.get(); // Different for every thread!
    }
    ...
```

המתודה getIdentifier תחזיר ערך שונה עבור כל חוט (בהנחה שהערך אותחל כראוי). ללא שימוש באוביקט ThreadLocal (למשל בעזרת משתנה סטטי יחיד) הערך היה משותף לכל החוטים.

קראו בעיון את התיעוד של המחלקה ThreadLocal המופיע לעיל וממשו את המחלקה בהתאם לדרישות.

רמזים:

- זכרו כי כדי לקבל מזהה של החוט הנוכחי ניתן להשתמש בפונקציה הסטטית Thread.currentThread() המחזירה אובייקט מטיפוס Thread המזהה את החוט הנוכחי (**ושונה מהחוטים האחרים**).

static Thread	**currentThread**() Returns a reference to the currently executing thread object.

- שימו לב שאין דרך לדעת מראש כמה חוטים ירוצו. עליכם לתמוך בפתרון שיתמוך **בכל מספר** של חוטים.

```
public class ThreadLocal<T> {
    private Map<Thread, T> localMap = new HashMap<Thread, T>();

    public ThreadLocal() {
    }

    public T get() {
        Thread t = Thread.currentThread();
        if (!localMap.containsKey(t)) {
            localMap.put(t, initialValue());
        }
        return localMap.get(t);
    }

    public void remove() {
        localMap.remove(Thread.currentThread());
    }

    public void set(T value) {
        localMap.put(Thread.currentThread(), value);
    }

    protected T initialValue() {
        return null;
    }
}
```

ג. **(20 נקודות)** נרצה להשתמש במנגנון שמירת ההודעות מסעיף א' כך שכל חוט יכתוב את הודעותיו ל-logger נפרד, לשם כך נשתמש במחלקה שמימשתם בסעיף ב'. נשתמש בתבנית העיצוב Factory כדי לייצר אובייקטים מטיפוס הממשק ILogger. אתם תממשו את המחלקה LoggerFactory המייצרת אובייקטים מטיפוס ILogger מחלקה זו תספק פונקציה סטטית יחידה getLogger().

```
public class LoggerFactory {
    ...
    public static ILogger getLogger() {
        ...
    }
}
```

הפונקציה getLogger() תחזיר אובייקט מטיפוס SimpleLogger שיאותחל עם מערך הודעות, המכיל לכל היותר 10 הודעות. הפונקציה תחזיר אובייקט שונה עבור כל חוט (כמובן שקריאות לפונקציה מאותו החוט יחזירו את אותו האובייקט).

השתמשו במחלקה ThreadLocal לצורך מימוש LoggerFactory. ספקו שני מימושים שונים: (1) בדיקה במתודה getLogger האם כבר בוצע אתחול לערך של המשתנה המתאים לחוט הנוכחי; (2) דריסה של המתודה initialValue (ראו סעיף ב') באמצעות מחלקה אנונימית.

מימוש 1:

```
public class LoggerFactory {
    private static final int DEFAULT_SIZE = 10;
    private static final ThreadLocal<ILogger> logger =
        new ThreadLocal<ILogger>();

    public static ILogger getLogger() {
        if (logger.get() == null) {
            logger.set(new SimpleLogger(DEFAULT_SIZE));
        }
        return logger.get();
    }

    private LoggerFactory1() {
    }
}
```

מימוש 2:

```
public class LoggerFactory {
    private static final int DEFAULT_SIZE = 10;
    private static ThreadLocal<ILogger> logger =
        new ThreadLocal<ILogger>() {
            @Override
            protected ILogger initialValue() {
                return new SimpleLogger(DEFAULT_SIZE);
            }
        };

    public static ILogger getLogger() {
        return logger.get();
    }

    private LoggerFactory() {
    }
}
```

שאלה 2 (20 נקודות)

מוגדר המנשק INode, המתאר קודקוד בעץ. בעזרת שירותי המנשק ניתן לקבל מכל קודקוד את התווית שלו, המכילה תיאור טקסטואלי של הקודקוד, ואת רשימת קודקודי הבנים שלו בעץ (שתהיה *null* במקרה של עלה בעץ).

```
public interface INode {
    /**
     * Returns the node's label - a textural description of the
     * node
     */
    public String getNodeLabel();

    /**
     * Returns a list containing all children nodes of the
     * current node in the tree. Returns null when called on
     * a leaf
     */
    public List<INode> getChildren();

}
```

א. **(10 נקודות)** כתבו מחלקה TreePrinter המכילה שירות סטטי אחד בשם printTree אשר מקבל קודקוד מטיפוס INode ומדפיס את התוויות של כל קודקודי העץ המושרש בקודקוד הנתון. כל תווית תודפס בשורה נפרדת. סדר ההדפסה אינו משנה כל עוד התוויות של כל הקודקודים בעץ הודפסו.

```java
public class TreePrinter {

    public static void printTree(INode root) {
        if (root == null) {
            return;
        }
        System.out.println(root.getLabel());
        if (root.getChildren() != null) {
            for (INode node : root.getChildren()) {
                printTree(node);
            }
        }
    }

}
```

ב. **(10 נקודות)** המחלקה FileSystemNode מממשת את המנשק INode. אובייקט מטיפוס FileSystemNode מתאר מסלול (path) לקובץ או לספרייה במערכת קבצים. עליכם לממש את המחלקה כך שתוכלו להשתמש ב-TreePrinter כדי להדפיס את התוכן של ספרייה (כולל כל תתי הספריות) במערכת הקבצים. עבור כל קובץ יודפס: "File: " + <filename>. ועבור כל ספריה יודפס: "Directory: " + <directory name>.

בנספח תמצאו תיעוד של המחלקה File.

```
public class FileSystemNode implements INode {
    private File file;

    public FileSystemNode(String path) {
        this(new File(path));
    }

    public FileSystemNode(File root) {
        if (!root.exists()) {
            throw new IllegalArgumentException("doesn't exist");
        }
        this.file = root;
    }

    @Override
    public String getLabel() {
        return (file.isDirectory() ? "Directory: " : "File: ")
                +  file.getName();
    }

    @Override
    public List<INode> getChildren() {
        List<INode> children = null;
        if (file.isDirectory()) {
            children = new ArrayList<INode>();
            for (File f : file.listFiles()) {
                children.add(new FileSystemNode(f));
            }
        }
        return children;
    }
}
```

שאלה 3 (20 נקודות)

א. **(5 נקודות)** בהינתן שירות המצהיר שייתכן שהוא זורק חריג נבדק (checked exception). מהן אפשרויות הטיפול בחריג העומדות בפנינו בקריאה לשירות?

1. הפונקציה הקוראת לשירות תצהיר שייתכן וגם היא זורקת את החריג
2. נעטוף את הקריאה לשירות ב try-catch ונטפל בחריג

ב. **(5 נקודות)** כאשר אנו משרשרים אובייקט למחרוזת הופך הקומפיילר את הפעולה לשרשור מחרוזות על ידי קריאה לשירות ()toString. הסבירו מדוע זה מהלך חוקי מצד הקומפיילר.

השירות toString מוגדר בטיפוס Object וכל הטיפוסים (הלא פרימיטיביים) בג'אווה יורשים מ-Object. לפיכך לכל טיפוס בג'אווה מוגדר השירות toString.

ג. **(5 נקודות)** נתונה חתימת המתודה foo: public void foo(List<?> list)
מהם הערכים שהמתודה foo יכולה להוסיף לרשימה שהיא מקבלת בתור ארגומנט. בחרו את התשובה הטובה ביותר.

1. כל ערך שהוא
2. null
3. לא ניתן להוסיף ערכים כלל
4. זהו תחביר לא חוקי, התכנית לא מתקמפלת

ד. **(5 נקודות)** מה יקרה כשננסה לקמפל ולהריץ את הקוד הבא:

```
public static void main(String[] args) {
    Set vals = new TreeSet();
    vals.add("one");
    vals.add(1);
    vals.add("two");
    System.out.println(vals);
}
```

1. הקוד לא מתקמפל
2. הקוד מתקמפל עם אזהרות ומדפיס [one, 1, two]
3. הקוד מתקמפל ללא אזהרות ומדפיס את הערכים בסדר כלשהו
4. הקוד מתקמפל עם אזהרות, בזמן ריצה נזרק חריג