קורס ג'אוה ומבוא לפיתוח ווב – תרגיל בנושא reflection

תוכן העניינים

[כללי 2](#_Toc480584011)

[מבנה התרגיל 2](#_Toc480584012)

[ניקוד 2](#_Toc480584013)

[איך בודקים ? 3](#_Toc480584014)

[מה מגישים ? 3](#_Toc480584015)

[נספח – מקרה הבדיקה לדוגמא 3](#_Toc480584016)

## כללי

**מועד הגשה: 31.7.20 צורת הגשה: ביחידים בלבד**

בתרגיל זה תזכו להכיר ולתרגל את מנגנון ה Reflection בג'אווה.

המצגת מכילה ידע בסיסי המהווה נק' פתיחה בלבד לנושא זה, ולכן כחלק מהתרגיל תדרשו גם ללמוד לבד נושאים נוספים הקשורים לעולם ה reflection שייתכן ואינם מכוסים (או אינם מכוסים כהלכה) במצגת.

שימו לב כי במצגת ישנם 3 נושאים. התרגיל הוא על הנושא הראשון בלבד של Reflection.

מהות התרגיל היא לכתוב investigator שיודע לקבל מופע (instance) של איזה שהוא אובייקט, ואז יודע "לחקור" אותו ולענות על מספר שאלות בהקשרו.

במהלך התרגיל תצטרכו להשתמש אך ורק ביכולת ה Reflection של ג'אווה כפי שמוסברות במצגת.

**אין להשתמש (ואין שום צורך) בשום ספריית צד שלישי כדי לבצע את המטלות השונות !**

צפי העבודה על התרגיל, בהינתן שקראתם והבנתם את המצגת, הוא 3-4 שעות.

צפי אורך הקוד שעליכם לכתוב הוא לא יותר מ 250 שורות (אני עשיתי זאת ב 160 שורות מרווחות היטיב..)

הבדיקה לתרגיל תבוצע בצורה אוטומטית, ע"י קוד שיטען את הקובץ שלכם ויריץ את סט הבדיקות שנגזרות ממנו, כלומר יקרא לכל השיטות המוגדרות ב interface שבחרתם, תוך השוואת הערך המוחזר מהקריאה אל ערך מצופה.

## מבנה התרגיל

התרגיל מכיל ממשק בשם Investigator, אותו עליכם לממש. מהות השיטות בממשק הוא לחקור instance של class אחר.

כל שיטה בממשק מתועדת היטיב מבחינת מה היא צריכה לעשות, מה הפרמטרים שהיא מקבלת ומה היא צריכה להחזיר.

חלק מהתרגיל כולל גם התמודדות עם התיעוד והבנה בעזרתו בלבד מה עליכם לעשות בכל שיטה ושיטה.

**הערות חשובות**:

1. שימו לב כי ה class שאתם מממשים חייב להכיל default public constructor !
2. שימו לב כי עליכם למקם את הממשק שקיבלתם (Investigator.java) בדיוק תחת ה package: **reflection.api** (ובהתאם לכך גם היררכיית הספריות כמובן).
3. במידה ויש מטודות הדורשות תפיסת exception, יש לתפוס אותו אולם אין להדפיסו ! (זה יוצר אי-סדר בהדפסות הפלט). במידה ומתרחשת תקלה או שתזרקו הלאה את ה exception ותוכנית הבדיקה תתמודד איתו בדרכה (הורדת ניקוד על הסעיף המדובר) או שתחזירו ערך כלשהוא כתוצאת המטודה (גם אם הוא לא נכון).
4. התוכנית מתחילה מקריאה לפונקיה load (אחת הפונקציות המתוארות בממשק). בפונקציה זו תקבלו את ה instance אותו עליכם לחקור. אפשר להניח כי זו הפונקציה הראשונה שתקרא, וכי היא תיקרא בדיוק פעם אחת.
5. כל מטודה בתרגיל עובדת בפני עצמה ופועלת על ה instance שקיבלתם במטודה load. אין להניח או להסתמך על סדר בקריאת המטודות (למען האמת בכל בדיקה הסדר הוא רנדומלי)

## ניקוד

התרגיל שווה עד 5 נקודות **מהציון הסופי** (!!)

(לא רע ל 3 - 4 שעות עבודה ו 200 שורות קוד...)

## 

## איך בודקים ?

קיבלתם את תוכנית הבדיקה עצמה, אותה אני הולך להפעיל גם בעצמי לטובת בדיקת התרגיל.

התוכנית מקבלת כפרמטר חיצוני את שם הקובץ המקומפל שלכם (.class) ובודקת אותו על מספר מקרי בדיקה.

בעותק התוכנית שקיבלתם כרגע, יש דוגמא למקרה בדיקה פשוט עליו תוכלו לנסות ולהתנסות ולוודא אם אתם בכיוון הנכון או אם לאו.

כדי להפעיל את תוכנית הבדיקה עליכם לנווט לספרייה בה נמצאת תוכנית הבדיקה להקליד ב CMD:

…\> RunTester <your .class file name>

אם הכל עובד כשורה, תקבלו את הפלט הבא:

Test name: Rectangle Class

Testing Rectangle Class basics...

Testing [ getTotalNumberOfMethods ]: expecting answer [6] and got [6]

…

Test Score: 100

## מה מגישים ?

עליכם להגיש קובץ zip, הכולל **בדיוק** 2 קבצים (ו 2 קבצים בלבד !):

1. קובץ התוכנית שלכם בלבד, בגרסתו המקומפלת (.class).
2. קובץ קוד המקור (למקרה של בעיות חמורות בלבד)

שם קובץ הזיפ צריך להכיל את שמכם ואת הת.ז. (אין להגיש קובץ readme בתרגיל זה...)

**מה לא מגישים ?**

1. את קובץ הממשק שקיבלתם
2. תיקיית פרויקט...
3. כל זבל אחר שבמקרה יושב לכם ליד הקוד...

## 

## מקרה בדיקה לדוגמא

במקרה זה מתואר האובייקט Rectangle היורש מאובייקט שנקרא Polygon.

מימוש זה כבר מוטמע בתוכנית הבדיקה שקיבלתם ומופע שלו יינתן כקלט לתוכנית שלכם.

המופע יאותחל בצורה הבאה:

**rectangle** = **new** Rectangle(4,6);

(המימוש הוא חלקי ולא תמיד הגיוני – אז אל תתפסו לקטנות...)

**public class** Polygon {  
  
 **private** Set<Point> **points**;  
  
 **public** Polygon() {  
 **points** = **new** HashSet<>();  
 }  
  
 **public int** getTotalPoints() {  
 **return points**.size();  
 }  
  
 **protected void** addPoint(**int** x, **int** y) {  
 **points**.add(**new** Point(x, y));  
 }  
}

**public class** Rectangle **extends** Polygon **implements** Comparable, Serializable {  
  
 **private int x**;  
 **private int y**;  
 **private final int SCALE** = 2;  
  
 **public static void** PRINT\_SOMETHING() {  
 System.***out***.println(**"this is a static method"**);  
 }  
  
 **public** Rectangle() {  
 **x** = -1;  
 **y** = -1;  
 }  
  
 **public** Rectangle(**int** x, **int** y) {  
 **this**.**x** = x;  
 **this**.**y** = y;  
 updateParent();  
 }  
  
 **private void** updateParent() {  
 addPoint(0, 0);  
 addPoint(**x**, 0);  
 addPoint(0, **y**);  
 addPoint(**x**, **y**);  
 }  
  
 **public int** calcArea() {  
 **return x** \* **y**;  
 }  
  
 **public int** calcPerimeter() {  
 **return** twice(**x**) + twice(**y**);  
 }  
  
 **private int** twice(**int** num) {  
 **return** 2 \* num;  
 }  
  
 @Override  
 **public int** compareTo(Object o) {  
  
 **return this**.calcArea() - ((Rectangle)o).calcArea();  
 }  
}