תרגיל בית 4 - היכרות עם Java

כללי

1. מועד ההגשה 22.6.23 בשעה 23:55.
2. מטרת התרגיל היא היכרות מעמיקה עם תכנות ב- Java, שימוש במנגנוני Reflection, Annotation.
3. אחראי על התרגיל: ג'וליאן מור. שאלות יש לשלוח למתרגל האחראי על התרגיל במייל julianmour@campus.technion.ac.il עם הנושא: "236703 HW4"
4. קראו היטב את ההוראות, במסמך זה ובקוד שניתן לכם. מומלץ לקרוא את **כל** התרגיל, מסמך הדוגמא והקוד הניתן לכם לפני שמתחילים לפתור, זה יחסוך לכם הרבה זמן ומאמץ בתכנון הפתרון, ובמימושו.
5. הקפידו על קוד ברור, קריא ומתועד ברמה סבירה. עליכם לתעד כל חלק שאינו טריוויאלי בקוד שלכם.
6. מהירות ביצוע אינה נושא מרכזי בתרגילי הבית בקורס. בכל מקרה של התלבטות בין פשטות לבין ביצועים, העדיפו את המימוש הפשוט.
7. הימנעו משכפול קוד והשתמשו במידת האפשר בקוד שכבר מימשתם.
8. בכדי להימנע מטעויות, אנא עיינו ברשימת ה FAQ המתפרסמת באתר באופן שוטף.

הקדמה

בתרגיל זה נפתח מסגרת קטנה עבור פיתוח מונחה-התנהגות. פיתוח מונחה-התנהגות (BDD – Behaviour-Driven Development) הינה מתודולוגיית פיתוח תוכנה המבוססת על TDD – Test-Driven Development. ב-TDD הרעיון הוא לכתוב את בדיקות היחידה בטרם נכתב הקוד אותן הן בודקות. את בדיקות היחידה כותבים באמצעות חבילת תוכנה המיועדת להרצה אוטומטית של בדיקות היחידה, כדוגמת JUnit.
בתרגיל זה אנו נפתח חבילת תוכנה קטנה אשר באמצעותה יהיה אפשר לפתח על פי שיטת BDD.
לקריאה נוספת והרחבה על שיטת הפיתוח:

TDD - http://en.wikipedia.org/wiki/Test-driven_development

BDD - http://en.wikipedia.org/wiki/Behavior-driven_development

הסבר קצר על התרגיל

בתרגיל זה נבנה מערכת, התקרא **StoryTester**, שתאפשר למשתמשים בה לכתוב בדיקות לקוד שלהם על ידי כתיבת "סיפורים". מומלץ לקרוא את מסמך הדוגמאות ביחד עם התרגיל כבר מעכשיו, זה יקל על הבנת התרגיל.

סיפור

סיפור מורכב ממספר משפטים. אנו נתמוך ב-3 סוגי משפטים בלבד, כאשר כל משפט מוגדר על ידי מילה שמורה משלו. לכל משפט המילה הראשונה בו היא אחת מן המילים השמורות. תחילה, נציג את המילים השמורות:

Given

מציינת משפט המגדיר את האובייקט אותו אנו יוצרים ואת מצבו. עליו נריץ את הבדיקות.

When

מציינת משפט המגדיר את הבדיקה, איזה תרחיש אנו בודקים.

Then

מציינת משפט המגדיר את התוצאה הצפויה לאחר ביצוע התרחיש הנבדק.

סיפור חוקי יוגדר כסיפור המורכב באופן הבא:

1. משפט ראשון – משפט Given בודד.
2. מספר כלשהו (גדול ממש מ-0) משפטי When.
3. מספר כלשהו (גדול ממש מ-0) משפטי Then.
4. סוף סיפור (אין סימון מיוחד) או חזרה לסעיף 2.

בתרגיל זה אנו נתעסק אך ורק בסיפורים חוקיים (כלומר לא יבדקו סיפורים המורכבים באופן שונה).

משפט יקרא חוקי אם המשפט שלו מורכב באופן הבא:

1. מילה שמורה אחת בלבד מבין שלוש המילים הנ"ל.
2. אוסף מילים (גדול ממש מ-0).
3. מילה אחרונה המייצגת פרמטר. פרמטר זה יכול להיות או מחרוזת או מספר שלם בלבד.

בתרגיל זה נעסוק אך ורק בסיפורים המורכבים ממשפטים חוקיים בלבד.

חלק א – הגדרת אנוטציות

עליכם להגדיר שלוש אנוטציות: Given, When, ו- Then. על כל אנוטציה להיות מוגדרת באמצעות המטה-אנוטציות Retention ו-Target המתאימות (על ההתאמה להיות **מדויקת**, בהתאם לשימוש באנוטציה).

Given

המטרה של Given היא לסמן מתודה המגדירה את האובייקט אותו אנו יוצרים ואת מצבו, עליו נריץ את הבדיקות. לאנוטציה זו תהיה תכונה אחת מטיפוס String בשם value. ערך המחרוזות תשמש לזיהוי איזה מתודה נריץ, ועם איזה פרמטר, בהינתן משפט מסיפור הבדיקה.

When

המטרה של When היא לסמן מתודה המגדירה את הבדיקה, איזה תרחיש אנו בודקים. לאנוטציה זו תהיה תכונה אחת מטיפוס String בשם value. ערך המחרוזות תשמש לזיהוי איזה מתודה נריץ, ועם איזה פרמטר, בהינתן משפט מסיפור הבדיקה.

Then

המטרה של Then היא לסמן מתודה המגדירה את התוצאה הצפויה לאחר ביצוע תרחישים מסוימים. לאנוטציה זו תהיה תכונה אחת מטיפוס String בשם value. ערך המחרוזות תשמש לזיהוי איזה מתודה נריץ, ועם איזה פרמטר, בהינתן משפט מסיפור הבדיקה.

חלק ב – מימוש מחלקת הבדיקות

מימוש

בתרגיל זה נשתמש ביכולות ה-Reflection אשר למדתם בתרגול וב-annotations. לכן מומלץ לחזור על תרגולים אלו שנית לפני מימוש התרגיל. בנוסף ישנה דוגמא מפורטת, שמומלץ לעבור עליה.
בכלל, מותר לכם להוסיף מחלקות מקוננות (ולא מקוננות) לצורך מימוש התרגיל. מומלץ להשתמש ב-enum בעת הצורך.
אתם רשאים להוסיף מחלקות ומתודות כרצונכם.

אלגוריתם הבדיקה

בהינתן סיפור, נפרק אותו לכל משפטיו. לכל משפט ניקח את המילה השמורה המייצגת אותו, ונחפש במחלקת הבדיקות מתודה אשר לה אנוטציה מתאימה. מבין המתודות בעלות האנוטציה המתאימה, נחפש את זו שהמשפט שלה מתאים למשפט המוגדר בסיפור. לאחר מכן, נגזור את הפרמטר המתאים מהמשפט בסיפור ונפעיל את המתודה שמצאנו עם הפרמטר שגזרנו.

התאמת משפט לאנוטציה:

משפט מורכב מ- $X_1 + Y_1 + Z_1$ כאשר X_1 הוא המילה שמורה Given או When או Then, Y_1 הוא אוסף מילים, ו-Z_1 הוא מילה המייצגת פרמטר. אנוטציה מורכבת מ- $@X_2(Y_2 + \&Z_2)$ כאשר X_2 הוא המילה שמורה Given או When או Then, Y_2 הוא אוסף מילים, ו-Z_2 הוא מילה המייצגת פרמטר.
אנוטציה מתאימה למשפט אם ורק אם $X_1 = X_2$ וגם $Y_1 = Y_2$.

טיפוס הפרמטר שנמצא (Z_1) בסוף כל משפט יכול להיות *Integer* או *String* בלבד.
כל משפט מכיל פרמטר אחד בדיוק (לא 0 ולא 2 ומעלה). לכן גם כל מתודה במחלקת הבדיקות יכולה לקבל פרמטר אחד בדיוק.
אם במשפט ניתן פרמטר (Z_1) מטיפוס *String*, אז מספר המילים בו שווה ל-1 (כלומר לא ניתן לספק מחרוזת עם רווחים/*Tab*-ים וכו' – מילה אחת בלבד).
אלו הן הנחות/דרישות מקלות לתרגיל.

עבור כתיבת מחלקת הבדיקות עבור התרגיל אנו נעזר במתודה *Assert.assertEquals(expected, actual)* בלבד. לא נשתמש ב-*asserts* נוספים בכלל (כלומר, לא נשתמש ב-*assertSame*, *assertNotNull* וכו').

המחלקה StoryTesterImpl

מממשת את הממשק StoryTester. במחלקה זו אתם נדרשים לממש שתי מתודות (אתם רשאים להוסיף עוד):

void testOnInheritanceTree(String story, Class<?> testClass)

מתודה זו מקבלת סיפור, על פי הפורמט שהגדרנו בתחילת המסמך, ומחלקת בדיקות.
המתודה יוצרת אובייקט של מחלקת **testClass**, עוברת על כל המשפטים בסיפור ומריצה את המתודות ממחלקת הבדיקות המתאימות למשפטים, באמצאות האובייקט שיצרנו. המתודות המתאימות יכולות להימצא **במחלקת הבדיקות הנוכחית** ובכל **מחלקת בדיקות** אחרת ממנה המחלקת בדיקות הנוכחית **יורשת**. כלומר, יכול להתקיים כי משפט Given מסוים לא מוגדר על אף מתודה במחלקה testClass, אך הוא מוגדר על מתודה במחלקה אחרת ממנה testClass יורש.
לכן, לכל משפט, יש לחפש את המתודה המתאימה (המתודה המתאימה היא מתודה שיש לה אנוטציה המתאימה למשפט) במעלה היררכיית הירושה, כאשר מתחילים מהמחלקה הנוכחית ועולים בכל פעם צעד אחד מעלה בהיררכיה. כאשר מוצאים מתודה מתאימה, מפעילים אותה עם הפרמטר של המשפט.
אם עבור משפט מסוים לא מוצאים מתודה מתאימה, יש לזרוק מיד את החריגה המתאימה לסוג המשפט.
החריגות הן GivenNotFoundException, ThenNotFoundException, WhenNotFoundException.
שימו לב שאתם יכולים להיעזר בחריגה WordNotFoundException אשר ממנה יורשות שאר החריגות שצוינו למעלה.

void testOnNestedClasses(String story, Class<?> testClass)

מתודה זו מקבלת סיפור, על פי הפורמט שהגדרנו בתחילת המסמך, ומחלקת בדיקות. את הסיפור יש להריץ או על המחלקה הנוכחית (testClass), או על אחת מן המחלקות המקוננות הנמצאות בה.
יש להריץ סיפור על מחלקה בתנאי והמשפט Given הראשון קיים על מתודה במחלקה או בהיררכיית הירושה שלה (גם כאן מדובר על מחלקת הבדיקות ומחלקות הבדיקות ממנה היא יורשת). לכן, אם המחלקה הנוכחית בעלת מתודה (אצלה או בהיררכיית הירושה) שעליה מוגדר משפט Given מתאים, יש להריץ את הסיפור עליה (בדומה למתודה הקודמת).
אם לא מוגדר משפט Given כזה במחלקה הנוכחית, יש לחפש מחלקה מקוננת, באופן רקורסיבי (כלומר לחפש גם בתוך מחלקה מקוננת שנמצאת בתוך מחלקה מקוננת וכן הלאה...), שבה כן מוגדרת מתודה עם המשפט המתאים (או בהיררכיית הירושה שלה). הנחה מקלה – תהיה קיימת רק מחלקה מקוננת בודדת אחת לכל היותר המתאימה למשפט ה-Given הראשון. אם נמצא מחלקה מקוננת מתאימה יש להריץ את הסיפור עליה (בדומה למתודה הקודמת).
במידה ואף מחלקה לא מתאימה צריך לזרוק את החריגה GivenNotFoundException.
את הסיפור מריצים בדיוק כמו במתודה הקודמת (כלומר ברגע שמצאתם את ה Given המתאים, צריך לחפש מתודות רק במחלקה שבה נמצא ה-Given או בהיררכיית הירושה שלה) ולכן גם זריקת החריגות זהה לגבי מתודה זו.
זכרו שכדי להפעיל מתודות של מחלקה מקוננת, יש צורך במופע של המחלקה (במקרה של מקוננת סטטית לא צריך, אך אתם עדיין רשאים ליצור מופע של המחלקה הסטטית). יתר על כן, מופע של מחלקה מקוננת קשור למופע של המחלקה העוטפת.

שוב, שימו לב, כאשר דובר על היררכיית ירושה במתודות הנ"ל, דובר על היררכיית ירושה אצל המחלקות הבודקות (testClass) ולא הנבדקות! בנוסף, ניתן להגדיר בכל מחלקת בדיקות מתודות שהן private או protected. גם אותן עליכם להריץ, כלומר יש להתעלם מה-modifier של המתודות בעת הפעלת המתודות ולהריץ את כולן.

הרצת סיפור עוברת בהצלחה במידה וכל משפטי ה-Then קיימו את המצופה מהן (לא זרקו חריגה). הרצת סיפור נכשלת במידה ואחד או יותר ממשפטי ה-Then לא מתקיים (נזרקת ממחלקת הבדיקות החריגה `ComparisonFailure`). אנו נרצה להריץ את כל הסיפור, כך שגם אם אחד ממשפטי ה-Then נכשל, נוכל להמשיך ולהריץ את כל שאר הסיפור, ולגלות כמה משפטים נכשלו. הבעיה היא שלא נרצה שמשפטי ה-When שלפני ה-Then שנכשל ישפיעו על המשך הסיפור, כי כנראה למתודות שהם הפעילו יש טעות. לכן לפני כל הפעלת סדרת משפטי When נעשה לאובייקט שיצרנו ממחלקת הבדיקות גיבוי (פרטים על הגיבוי בהמשך), ואם משפט Then נכשל, אז נשחזר את האובייקט. במידה והרצת סיפור נכשלת יש לזרוק **בסוף** הרצת הסיפור חריגה מסוג StoryTestException שתייצג את הכישלון הראשון (ה-Then הראשון שנכשל) ואת מספר הכישלונות הכללי של הסיפור:

StoryTestExceptionImpl

מממשת את המחלקה האבסטרקטית StoryTestException. חריגה זו נזרקת כאשר הרצת סיפור נכשלת. חריגה זו תגדיר את המתודות:

String getSentance()

מתודה זו מחזירה את המשפט כולו (כולל המילה השמורה והפרמטר בסוף – המשפט שקיבלנו מתוך הסיפור בדיוק) – **ללא התו '\n' בסופו.**

String getStoryExpected()

מתודה זו מחזירה מחרוזת שמייצגת את הערך של הפרמטר במשפט אשר לקוח מתוך הסיפור.

String getTestResult()

מתודה זו מחזירה את המחרוזת שמייצגת את הפרמטר שבאמת התקבל על ידי הרצת המחלקה הבודקת.

Int getNumFail()

מתודה זו מחזירה את מספר משפטי ה-Then שנכשלו בזמן הרצת הסיפור.

גיבוי מצב האובייקט ושחזורו במידת הצורך

נבצע גיבוי לשדות האובייקט לפני כל הפעלת סדרת משפטי When ונשחזר במידת הצורך. לשם הפשטות, הגיבוי צריך להתבצע על השדות של מחלקת אובייקט היעד בלבד (האובייקט שיצרתם), ולא על השדות של המחלקות במעלה שרשרת הירושה. לא צריך לגבות את המחלקות המקוננות, או שדותיהם. כדי להגדיל את הסיכוי שהגיבוי יחזיק ערכים שלא יושפעו מהפעלת המתודות, עליו להתבצע לפי סדר העדיפויות הבא:

1. אם האובייקט שבשדה תומך ב-clone, הגיבוי ישמור שכפול של אותו האובייקט.
2. אם לאובייקט שבשדה יש copy constructor (בנאי שמקבל אובייקט מאותו הטיפוס), יעשה בו שימוש כדי ליצור אובייקט חדש מאותו הטיפוס, והוא זה שישמר בגיבוי.
3. אם שתי האפשרויות הקודמות לא קיימות, יישמר בגיבוי האובייקט שבשדה עצמו.

לדוגמא, אם באובייקט שעליו מפעילים את המתודה הנבדקת יש שדה מטיפוס java.util.Date (שמממש את Cloneable), נשמור בגיבוי עותק שלו שניצור באמצעות קריאה ל-clone. אם יהיה שדה מטיפוס String, נשתמש בבנאי שלו שמקבל String.

התוצאה של שחזור הגיבוי היא שכל השדות שעומדים בשני התנאים הראשונים יצביעו על אובייקטים שונים מאלה שהצביעו לפני הרצת המתודה, אבל ה-state של האובייקטים המוצבעים לפני ואחרי יהיה זהה.

הערות נוספות

- ניתן להניח שהמתודות המוגדרות במחלקות הנבדקות ובמחלקות הבודקות אינן זורקות חריגות. החריגה היחידה שיכולה להיזרק ממחלקה בודקת היא `ComparisonFailure`.
- הנחה מקלה - למחלקות הבדיקה (גם המקוננות) יש בנאי ברירת מחדל חסר פרמטרים בלבד.
- אם אחד מהארגומנטים שניתנו לפונקציה (**testOnNestedClasses** או **testOnInheritanceTree**) הוא null אז יש לזרוק מיד IllegalArgumentException. אם שני הארגומנטים לא null אז מובטח שהם ארגומנטים חוקים לפונקציה.
- אם לא קיימת אנוטציה מתאימה למשפט בסיפור יש לזרוק מיד את השגיאה המתאימה, ולא לחכות לסוף הסיפור.
- מסופק לכם טסט אחד מינימלי מאוד. **מומלץ מאוד** לעבור אותו לפני שמגישים. הגשה שתכשל בטסט הזה, כנראה תכשל בכל שאר הטסטים.

דרישות

- כל המחלקות שלכם צריכות להיות ממומשות ב: package solution.
- אין להדפיס לפלט הסטנדרטי או לפלט השגיאות הסטנדרטי. אם אתם משתמשים בפלט לצורך בדיקות, הקפידו להסיר את ההדפסות לפני ההגשה.
- אין לשנות את הקבצים המצורפים (של package provided). הבודק האוטומטי דורס את הקבצים ע"י הגרסה המצורפת.

אין להגיש את הטסטים שלכם!

הוראות הגשה

- בקשות לדחייה יש לשלוח למתרגל האחראי על הקורס (ג'וליאן) **בלבד**. מכיוון שבקורס יש מדיניות איחורים – ראו מידע באתר – דחיות יאושרו רק מסיבות לא צפויות או לא נשלטות (כמו מילואים).
- הגשת החלק הרטוב תתבצע אלקטרונית בלבד, יש לשמור את אישור השליחה!
- יש להגיש קובץ בשם OOP4_<ID1>_<ID2>.zip המכיל:
  - קובץ בשם readme.txt המכיל שם, מספר זיהוי וכתובת דואר אלקטרוני עבור כל אחד מהמגישים.
  - על ה- zip להכיל את כל קבצי הקוד שכתבתם לצורך התרגיל, בלי תיקיות נוספות ב- zip .
- אין להגיש את הקבצים המצורפים לתרגיל ואין להגיש test ים.
- הגשה שלא לפי ההוראות תגרור הורדת ציון בהתאם.