תרגיל 3: Java מתקדם

כללי

1. מועד הגשה 25/8/19 בשעה 23:55
2. מטרת התרגיל היא היכרות מעמיקה עם תכנות ב-Java, שימוש במנגנוני Reflection, ו-Annotations.
3. אחראי על התרגיל: נתן בגרוב. שאלות יש לשלוח במייל natanb@cs עם הנושא: **"236703 HW3"**.
4. שאלות בנושאים אדמיניסטרטיביים (כמו מילואים) יש לשלוח למתרגל האחראי על הקורס, אופיר, תחת אותה כותרת.
5. קראו היטב את ההוראות, במסמך זה ובקוד שניתן לכם. **מומלץ לקרוא את כל התרגיל לפני שמתחילים לפתור, זה יחסוך לכם הרבה זמן ומאמץ בתכנון הפתרון ובמימושו.**
6. תרגיל זה ארוך יותר מהתרגילים הקודמים והוא מצריך יותר זמן, לכן מומלץ להתחיל אותו מוקדם. בנוסף, מומלץ לקרוא אותו יותר מפעם אחת.
7. הקפידו על קוד ברור, קריא ומתועד ברמה סבירה. עליכם לתעד כל חלק שאינו טריוויאלי בקוד שלכם.
8. מהירות ביצוע אינה נושא מרכזי בתרגילי הבית בקורס. בכל מקרה של התלבטות בין פשטות לבין ביצועים, העדיפו את המימוש הפשוט.
9. הימנעו משכפול קוד והשתמשו במידת האפשר בקוד שכבר מימשתם.
10. כדי להימנע מטעויות, אנא עיינו ברשימת ה FAQ המתפרסמת באתר באופן שוטף.

תרגיל זה הינו ארוך ומורכב. אנו ממליצים לכם לקרוא את כל התרגיל לפני תחילת המימוש ולהבין את הדוגמאות לעומק.

הקדמה

בתרגיל זה נפתח מסגרת קטנה עבור פיתוח מונחה-התנהגות. פיתוח מונחה-התנהגות, Behaviour Driven Development, היא מתודולוגיית פיתוח תוכנה המבוסס על Test Driven Development.

ב- TDD הרעיון הוא לכתוב את בדיקות היחידה (טסטים) בטרם נכתב הקוד אותן הן בודקות. אך, בשונה מ- TDD, בדיקות היחידה אינן יכתבו בשפת תכנות, אלא נכתבים בשפה המזכירה שפה טבעית (כמו אנגלית). טסטים כאלו נקראים סיפורים (stories).
סיפורי BDD יכולים להיות שימושיים מכיוון ש:

- הם טסטים שניתן לכתוב אותם ללא תלות בשפת תכנות מסוימת.
- הם מובנים לקריאה גם לכאלו שאינם יודעים לקרוא\לכתוב שפות תכנות (ולכן גם אנשים שלא באים מרקע של תוכנה יכולים להגדיר בעזרתם התנהגות רצויה).

בתרגיל זה אנו נפתח חבילת תוכנה קטנה אשר באמצעותה יהיה אפשר לפתח על פי שיטת BDD. כלומר למעשה המטרה היא לבנות מערכת היודעת לקבל סיפור, ולהריץ את הטסטים המתאימים לסיפור הזה.

לקריאה נוספת והרחבה על שיטת הפיתוח:
TDD - http://en.wikipedia.org/wiki/Test-driven_development
BDD - http://en.wikipedia.org/wiki/Behavior-driven_development

לדוגמה, נניח שאנו רוצים לכתוב מחלקה של כיתת לימוד, שיש לה מתודות של הגדרת כמות התלמידים בה, הגדרת כמות הכסאות השבורים בה, וקבלת מחרוזת המייצגת האם הכיתה מלאה או לא. אך, לפי מתודולוגיית הפיתוח, תחילה אנו נכתוב את בדיקת היחידה למחלקה ורק לאחר מכן את המחלקה עצמה. כלומר נכתוב את ה"שלד" של המחלקה שלנו, אך ללא מימוש של הפונקציות:

```
public class Classroom {
    //TODO
    public Classroom(int capacity) {
        //TODO
    }

    public void addStudents(int numberOfStudents) {
        //TODO
    }

    public void addBrokenChairs(int numberOfBrokenChairs) {
        //TODO
    }
```

```
    public String classroomCondition() {
        //TODO
        return null;
    }
}
```

לאחר מכן, נצטרך לרשום את הטסטים למחלקה (כמו לדוגמה, טסט שבודק שאם בכיתה יש מקום ל-40 אנשים, ויש בה 45 אנשים, אז היא במצב "מלא"). כמובן שהטסטים הללו לא יעברו, כי המחלקה ריקה. כעת, נכתוב בתוך מתודות המחלקה את הקוד הקצר ביותר שניתן כדי שהטסטים שלנו יעברו. את הטסטים לא נכתוב ב-Java כאמור, אלא בתור "סיפור".

הגדרות והסבר קצר על התרגיל

בתרגיל זה נבנה מערכת, התקרא **StoryTester**, שתאפשר למשתמשים בה לכתוב בדיקות לקוד שלהם על ידי כתיבת "סיפורים".

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.

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.

נגדיר זאת ביתר פירוט ונציג כמה הגדרות להמשך התרגיל:

<u>תת-משפט מסוג I</u> יקרא <u>חוקי</u> אם הוא מורכב באופן הבא:

1. אוסף מילים (גדול ממש מ-0) המופרדים ברווח. **המילה and והמילה or <u>אינן</u> יכולות להיות כחלק מאוסף המילים הנ"ל.**
2. **מילה <u>אחרונה</u> המייצגת פרמטר**. פרמטר זה יכול להיות או מחרוזת או מספר שלם בלבד. (מספר יכיל אך ורק את הספרות 0 עד 9 עם אפשרות לסימן מינוס (-) בהתחלה. פרמטר שאינו מספר הוא מחרוזת).

תת-משפט מסוג II יקרא חוקי אם הוא מורכב באופן הבא:

1. אוסף תתי-משפטים מסוג I חוקיים (גדול ממש מ-0) המופרדים על ידי המילה **and** (עם רווח לפניה ואחריה).

שני תתי-משפטים מסוג II יקראו שקולים אם הם זהים לחלוטין עד כדי הפרמטרים שהדרישה היחידה עליהם זה שהם צריכים להיות עם אותם **טיפוסי** פרמטרים בהתאמה.

לדוגמא:

the number of students in the classroom is 80 and the name of the classroom is Taub3

the number of students in the classroom is 140 and the name of the classroom is Ullman501

2 תת-המשפטים הנ"ל הם שקולים כי הם זהים לחלוטין חוץ מ-2 הפרמטרים שלהם, אך הפרמטרים שלהם עדיין שומרים על אותו סדר של טיפוסים. כלומר, אצל 2 המשפטים הפרמטר הראשון הוא מטיפוס מספר שלם והפרמטר השני הוא מטיפוס של מחרוזת.

משפט יקרא חוקי אם הוא בנוי באחת משתי הדרכים הבאות:

דרך א':

1. מילה שמורה אחת בלבד מבין שני המילים Given (יקרא משפט Given) ו- When (יקרא משפט When) (ואחריה רווח) .
2. תת-משפט מסוג II אחד בלבד.

או

דרך ב':

1. המילה השמורה Then (יקרא משפט Then) (ואחריה רווח).
2. אוסף תתי-משפטים מסוג II חוקיים **ושקולים** (גדול ממש מ-0) המופרדים על ידי המילה **or** (עם רווח לפניה ואחריה).

בתרגיל זה נעסוק אך ורק בסיפורים המורכבים ממשפטים חוקיים בלבד.
בסיפור חוקי, המפריד (delimiter) בין משפטים יהיה התו '\n'.

דוגמאות למשפטים חוקיים:

Given a classroom with a capacity of 75

כאשר המילה השמורה בדוגמה היא Given והפרמטר הוא המספר השלם 75

Then the number of students in the classroom is 80 and the number among them that are standing is 10 or the number of students in the classroom is 100 and the number among them that are standing is 50

כאשר המילה השמורה בדוגמה היא Then, ויש לה 2 תתי-משפטים מסוג II (מופרדים ע"י המילה or סוכן שקולים): אחד עם הפרמטרים 80 ו-10 והשני עם הפרמטרים 100 ו-50 בהתאמה.

סיפור חוקי מורכב באופן הבא:

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

דוגמאות לסיפורים חוקיים:

1.
Given a classroom with a capacity of 75
When the number of students in the classroom is 60
Then the classroom is not-full

2.
Given a classroom with a capacity of 50
When the number of students in the classroom is 40
When the number of broken chairs in the classroom is 10
Then the classroom is full
When the number of students in the classroom is 40
Then the classroom is not-full

3.
Given a classroom that the number of seats in it is 33
When the number of students in the classroom is 42 and the number among them that are standing is 6 (זה משפט אחד)
Then the classroom is not-full and quiet or the classroom is full and noisy

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

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 המחרוזת ישמש לזיהוי איזה מתודה נריץ, ועם אילו פרמטר/ים, 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 המחרוזת ישמש לזיהוי איזה מתודה נריץ, ועם אילו פרמטר/ים, 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 המחרוזת ישמש לזיהוי איזה מתודה נריץ, ועם אילו פרמטר/ים, בהינתן משפט מסיפור הבדיקה.

חלק ב - מימוש מחלקת הבדיקות

מימוש

בתרגיל זה נשתמש ביכולות ה-Reflection (אשר למדתם בתרגול) וב-annotations. מומלץ לחזור על תרגולים אלו לפני מימוש התרגיל. בנוסף ישנה דוגמא מפורטת, שמומלץ לעבור עליה.
בכלל, מותר לכם ואפילו מומלץ להוסיף מחלקות (מקוננות או לא מקוננות) ומתודות כרצונכם לצורך מימוש התרגיל. בנוסף, מומלץ להשתמש בכלים שנלמדו בכיתה. מומלץ להשתמש ב-enums בעת הצורך.

אלגוריתם הבדיקה

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

התאמת משפט לאנוטציה

משפט המורכב מ- n תתי-משפטים מסוג I ו- m תתי משפטים מסוג II (כמובן שעבור משפטי Given או When מתקיים m=1 כפי שהגדרנו קודם) נראה מהצורה:

$$X_1\ Y_1\ Z^1{}_1\ and\ Y_2\ Z^1{}_2\ and\ ...\ and\ Y_n\ Z^1{}_n\ or\ ...\ or\ Y_1\ Z^m{}_1\ and\ Y_2\ Z^m{}_2\ and\ ...\ and\ Y_n\ Z^m{}_n$$

כאשר:

- X_1 הוא מילה שמורה (Given, When או Then)
- Y_i הוא אוסף מילים
- Z_i^j הוא מילה המייצגת פרמטר (ה-j מייצגת את האינדקס של תת-המשפט מסוג II במשפט, וה-i מייצגת את האינדקס של התת-מילה מסוג I בתוך התת-משפט מסוג II שלה).
- שימו לב: בכל תת משפט מסוג II, כל ה-Y-ים זהים (בדומה למה שהגדרנו קודם, כי הם שקולים) אך ה-Z-ים אינם בהכרח זהים בין תתי-משפטים מסוג II (כי הפרמטרים לא צריכים להיות זהים בין תתי-משפטים מסוג II שקולים)

אנוטציה נראית מהצורה: $@x_1(y_1\ \&z_1\ and\ y_2\ \&z_2\ and\ ...\ and\ y_n\ \&z_n)$ כאשר:

- x_1 הוא מילה שמורה (Given, When או Then)
- y_i הוא אוסף מילים (המילה or אינה יכולה להיות כחלק מאוסף המילים הזה)
- z_i הוא מילה המייצגת פרמטר (כאשר התו & מגיע לפניו)

אנוטציה מתאימה למשפט אם ורק אם $X_1 = x_1$ **וגם** $Y_i = y_i$ **(לכל** $1 \le i \le n$**).**

טיפוס הפרמטר (Z_i) הנמצא בסוף כל משפט יכול להיות *Integer* או *String* בלבד.
אין חובה שיהיה יותר מתת-משפט אחד (מכל סוג), כלומר, משפט יכול להיות גם מהצורה $X_1\ Y_1\ Z_1$, והאנוטציה המתאימה לו תהיה מהצורה $@X_1(Y_1\ \&z_1)$.

<u>דוגמאות:</u>

1. המשפט

When the number of students in the classroom is 60

מתאים לאנוטציה

```
@When("the number of students in the classroom is &size")
```

2. המשפט

When the number of students in the classroom is 80 and the number among them that are standing is 10

מתאים לאנוטציה

```
@When("the number of students in the classroom is &students and the number among them that are standing is &standing")
```

3. המשפט

Then the classroom is not-full and not-empty or the classroom is almost-full and not-empty

מתאים לאנוטציה

```
@Then("the classroom is &condition and &condition")
```

<u>הנחות מקלות:</u>

1. מובטח שהאנוטציות ושהסיפורים מתאימים למתודות מבחינת מספר הפרמטרים והטיפוס שלהם. כלומר, מתודה תקבל את אותו מספר הפרמטרים כמו באנוטציה שלה, לפי אותו הסדר שהם מופיעים באנוטציה, וטיפוס הערכים של הפרמטרים הללו (המופיעים בסיפור) יתאימו לטיפוס הפרמטרים שהמתודה מקבלת. לדוגמה, מתודה f עם החתימה הבאה:

```
f(Integer x, String y)
```

עבור כל אנוטציה עבור הפונקציה f מובטח שהיא תהיה עם 2 פרמטרים בדיוק. כלומר האנוטציה הבאה היא אנוטציה אפשרית:

```
@Then("the number of available chairs in the classroom is &number and the name of the classroom is &string")
```

בנוסף מובטח שכל משפט המתאים לאנוטציה הנ"ל יהיה עם מספר בפרמטר הראשון ומחרוזת בפרמטר השני. כלומר המשפט הבאה הוא אפשרי:

Then the number of available chairs in the classroom is 40 and the name of the classroom is Taub9.

שימו לב, שעבור משפטי Then, התאמת הטיפוסים והכמות שלהם יהיה נכון לכל תת-משפט מסוג II. זאת אומרת שבדוגמה הזו (עם המתודה f) כל תת-טיפוס מסוג II של משפט Then שמתאים לפונקציה f צריך להכיל 2 פרמטרים בדיוק, כאשר הראשון מספר שלם והשני מחרוזת. כלומר המשפט הבא גם מתאים לפונקציה f:

Then the number of available chairs in the classroom is 40 and the name of the classroom is Taub9 or the number of available chairs in the classroom is 60 and the name of the classroom is Ulman200

2. אם במשפט ניתן פרמטר (Z_i) מטיפוס *String*, אז מספר המילים בו שווה ל-1 (כלומר לא ניתן לספק מחרוזת עם רווחים/*Tab*-ים וכו', אלא מילה אחת בלבד).
3. עבור כתיבת מחלקת הבדיקות עבור התרגיל, הטסטים של הסגל יעזרו במתודה `Assert.assertEquals(expected, actual)` על מחרוזות בלבד. לא נשתמש ב-*asserts* נוספים בכלל (כלומר, לא נשתמש ב-*assertNotNull*, *assertSame* וכו'). כלומר, **ניתן להניח שכאשר מתודה במחלקת הבדיקות זורקת חריגה, היא מטיפוס `ComparisonFailure`.**
4. למתודה תהיה לכל היותר אנוטציה אחת שמתויגת אליה.
5. רק מתודות המתויגות באנוטציה של Then יזרקו חריגת `ComparisonFailure`.

הממשק StoryTester

עליכם ליצור מחלקה בשם **StoryTesterImpl** אשר תממש את הממשק StoryTester. אתם נדרשים לממש את שתי מתודות הנמצאות בממשק (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, באמצעות האובייקט שיצרנו.
- המתודות המתאימות יכולות להימצא **במחלקת הבדיקות הנוכחית** ובכל **מחלקת בדיקות** אחרת ממנה מחלקת הבדיקות 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 (זו 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 את החריגה המתאימה לסוג המשפט. החריגות האפשריות הן:
  GivenNotFoundException, ThenNotFoundException, WhenNotFoundException.
  אתם יכולים להיעזר בחריגה WordNotFoundException אשר ממנה יורשות שאר החריגות שצוינו למעלה.

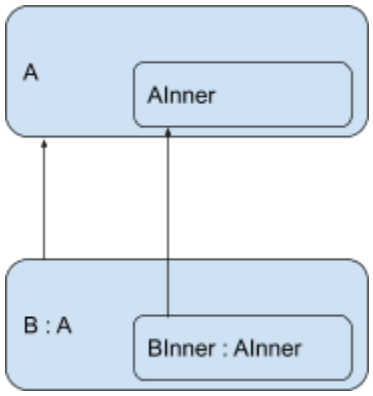
void testOnNestedClasses(String story, Class<?> testClass)

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:
  - תחילה, יש להריץ סיפור על מחלקה בתנאי והמשפט Given 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 או במעלה היררכיית הירושה שלך, אז 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: 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).

- 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 במקרה זה **זהה** כמו במתודה הקודמת. *ראו דוגמא למטה.
- אם אף מחלקה לא מתאימה - יש לזרוק את החריגה `GivenNotFoundException`.

דוגמא:

נניח ש- `testClass=B`. תחילה: אנו נחפש את המתודה בעלת האנוטציה `@Given` ב- B ואם לא מצאנו- נחפש ב- A. אחר כך: אם לא מצאנו- נחפש ב- `BInner` ואם לא מצאנו- נחפש ב- `AInner`. במידה והאנוטציה נמצאה ב- B או ב- A, אנו מריצים את שאר הטסט על **B** (!). במידה והאנוטציה נמצאה ב- `BInner` או ב- `AInner`, אנו מריצים את שאר הטסט על **`BInner`** (!). ניתן לחשוב על זה שעובדים עם `testClass` ומחליפים אותו למחלקה מקוננת במידה ולא מצאנו בהיררכית הירושה שלו, ואז קוראים רקורסיבית.	

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 (כפי שנלמד בתרגול).

<u>בנוגע ל-2 המתודות:</u>

- הרצת משפט **Then** <u>תחשב כהצלחה</u> כאשר **<u>לפחות אחד</u>** מתת-המשפטים מסוג II שלו עברו בהצלחה (לא נזרקה חריגה בהרצה שלהם). כלומר, משפט Then עם 3 תתי-משפטים מסוג II ידרוש עד 3 הרצות של המתודה המתאימה (כל פעם עם הפרמטרים המתאימים) ואם לפחות אחת מההרצות הנ"ל עוברת בהצלחה (לא נזרקת חריגה) אז הרצת משפט ה-Then יחשב כהצלחה, אחרת ייחשב ככישלון.
  - **שימו לב שאם תת-משפט מסוג II עובר בהצלחה, אז אין להריץ את שאר תתי-המשפטים מסוג II של אותו משפט Then.** (לדוג' במשפט Then עם 3 תתי-משפטים מסוג II כאשר הראשון לא עובר והשני עובר, אז נריץ את תתי-המשפט הראשון והשני אך **<u>לא</u>** את השלישי).
- הצלחה או כישלון של הרצת **<u>סיפור</u>** מוגדרת כך:
  - הרצת סיפור עוברת בהצלחה במידה וכל הרצות משפטי ה-Then הצליחו.
  - הרצת סיפור נכשלת במידה ואחת או יותר מהרצות משפטי ה-Then נכשלו.
  - **שימו לב שאפילו אם משפט Then נכשל, עדיין יש להמשיך להריץ את כל הסיפור.**

- 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.
- שוב, שימו לב, כאשר דובר על היררכיית ירושה במתודות הנ"ל, דובר על היררכיית ירושה אצל המחלקות הבודקות (testClass) ולא הנבדקות! כלומר, אם יש לנו מחלקה נבדקת Classroom ומחלקת בדיקות (מחלקה בודקת) ClassroomStoryTests, מדובר על ההיררכייה ב- ClassroomStoryTest. 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.
- ניתן להניח שהמתודות המוגדרות במחלקות הנבדקות ובמחלקות הבודקות אינן זורקות חריגות. החריגה היחידה שיכולה להיזרק ממחלקה בודקת היא ComparisonFailure.
- אם לא קיימת אנוטציה מתאימה למשפט בסיפור יש לזרוק מיד את השגיאה המתאימה, ולא לחכות לסוף הסיפור.
- הנחה מקלה - למחלקות הבדיקה (גם המקוננות) יש בנאי ברירת מחדל חסר פרמטרים בלבד.
- עבור משפט מסוים, לא ימצאו 2 אנוטציות מתאימות באותה מחלקה.
- אם במעלה הירושה יש יותר ממתודה אחד המתאימה למשפט מסוים, אז יש להריץ את המתודה הראשונה שאתם נתקלים בה כאשר אתם עולים במעלה הירושה (ורק אותה). כלומר, המתודה שהכי קרובה למחלקה שממנה התחיל החיפוש.

כישלון של משפטי Then וגיבוי

אנו נרצה להריץ את כל הסיפור, כך שגם אם אחד ממשפטי ה-Then נכשל (ונזכיר שכשלון של משפט Then מתרחש כאשר **כל** תתי-המשפטים מסוג II שלו נכשלו, כלומר כל ה-or-ים שלו נכשלו), נוכל להמשיך ולהריץ את כל שאר הסיפור, ולגלות כמה משפטים נכשלו. **שימו לב - לא 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.

המחלקה StoryTestExceptionImpl

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

String getSentence()

מתודה זו מחזירה את כל משפט ה-Then הראשון שנכשל (כולל המילה השמורה והפרמטר/ים – כלומר בדיוק המשפט שקיבלנו מתוך הסיפור) – **ללא התו '\n' בסופו.**

List<String> getStoryExpected()

מתודה זו מחזירה ,עבור ה-Then הראשון שנכשל, רשימה של מחרוזות. המחרוזת ה-i ברשימה זו מייצגת את הערך של הפרמטר (אשר לקוח מתוך הסיפור) שהוא זה שנכשל לפי חריגת ה-ComparisonFailure בתת-משפט מסוג ll ה-i . כלומר, את ה- expected בחריגה ComparisonFailure.

List<String> getTestResult()

מתודה זו מחזירה ,עבור ה-Then הראשון שנכשל, רשימה של מחרוזות. המחרוזת ה-i ברשימה מייצגת את הערך שבאמת התקבל (שאי ההתאמה שלו לערך המצופה הוא זה שהביא לחריגה להזרק) לפי חריגת ה-ComparisonFailure בתת-משפט מסוג ll ה-i . כלומר, את ה- actual בחריגה ComparisonFailure.

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 שדותיהן. 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. כלומר, צריך לבדוק שהמחלקה של השדה מממשת את Cloneable. אם היא אכן מממשת את Cloneable, אז ניתן להניח שהיא גם דורסת את המתודה clone ושאתם יכולים להשתמש במתודה הזאת כדי לשכפל את 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 (כלומר, לא העתק שלו, אלא השמה באמצעות =).

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.
לדוגמא, עבור המחלקה הבאה שנרצה לעשות עבורה גיבוי:

```
public class ClassroomStoryTest {
   private java.util.Date reconstructionTime;
   private String nameOfClassroom;
   private Chairs chairs;


   //...
}
```

הגיבוי שנעשה לכל שדה הוא:

- לשדה reconstructionTime מטיפוס java.util.Date (שמממש את Cloneable), נשמור בגיבוי עותק שלו שניצור באמצעות קריאה ל-clone.
- לשדה nameOfClassroom מטיפוס String, נשתמש בבנאי שלו שמקבל String.
- לשדה chairs מטיפוס Chairs (שזו מחלקה שיצרנו שאין לה copy constructor והיא אינה מממשת cloneable) אנחנו פשוט נשמור את האובייקט כמו שהוא.

דוגמאות נוספות

נמשיך את הדוגמא של מחלקת כיתות הלימוד. נכתוב מחלקת בדיקות:

```
public class ClassroomStoryTest {
   protected Classroom classroom;
   @Given("a classroom with a capacity of &capacity")
   public void aClassroom(Integer capacity) {
       classroom = new Classroom(capacity);
   }

   @When("the number of students in the classroom is &size")
   public void classroomIsWIthStudents(Integer size) {
      classroom.numberOfStudents(size);
   }

   @Then("the classroom is &condition")
   public void theClassroomCondition(String condition) {
       Assert.assertEquals(condition, classroom.classroomCondition());
   }
}
```

ניתן לראות שכל אנוטציה מקבלת מחרוזת. בסוף כל מחרוזת המילה האחרונה מייצגת פרמטר (אותו פרמטר שהגדרנו עבורו ערך במשפט בסיפור). פרמטר מסומן על ידי התו & בראשיתו. שם הפרמטר כשם המשתנה שהמתודה מקבלת. כל מחרוזת כזו תותאם למשפט חוקי בסיפור.

עבור כל סיפור, יש להריץ לפני כל רצף של When גיבוי לאובייקט של מחלקת הבדיקה. כלומר, אם הסיפור היה כתוב כך:

Given a classroom with a capacity of 75
When the number of students in the classroom is 60
Then the classroom is not-full
When the number of students in the classroom is 80
Then the classroom is full

סדר הפעולות שנעשה הוא:

- Create object `TEST` of class `ClassroomStoryTest`
- `Invoke aClassroom(75)`
- BackUp object `TEST`

- `Invoke classroomIsWIthStudents(65)`
- `Invoke  theClassroomCondition(not-full)`
- if *`assertEquals` fails and throws* `ComparisonFailure` restore object `TEST`
- BackUp object `TEST`
- `Invoke classroomIsWIthStudents(80)`
- `theClassroomCondition(full)`
- if *`assertEquals` fails and throws* `ComparisonFailure` restore object `TEST`
- if any Then sentence has failed throw `StoryTestException`

כעת, ניקח מחלקת בדיקות יותר מסובכת:

```
public class ClassroomStoryDerivedTest extends ClassroomStoryTest {
   @When("the number of broken chairs in the classroom is &broken")
   private void classroomIsWIthBrokenChairs(Integer broken) {
       classroom.brokenChairs(broken);
   }

   public class InnerClass extends ClassroomStoryTest {
       @Given("a classroom that the number of seats in it is &seats")
       public void aClassroom(Integer seats) {
           classroom = new Classroom(seats);
       }

      @When("the number of students in the classroom is &students and the
      number among them that are standing is &standing")
      private void classroomIsWIthStandingStudents(Integer students,
                                                   Integer standing){
       classroom.numberOfStudents(students - standing);
      }
   }
}
```

וניקח את הסיפור:

Given a classroom with a capacity of 50
When the number of students in the classroom is 40
When the number of broken chairs in the classroom is 10

Then the classroom is full

ונריץ את הסיפור באמצעות המתודה **testOnNestedClasses** עם מחלקת הטסט `ClassroomStoryDerivedTest`, סדר הפעולות שנעשה יהיה כך:

- Create object `TEST` of class `ClassroomStoryDerivedTest`
- `Invoke ClassroomStoryTest::aClassroom(50)`
- BackUp object `TEST`
- `Invoke ClassroomStoryTest::classroomIsWIthStudents(40)`
- `Invoke ClassroomStoryDerivedTest::classroomIsWIthBrokenChairs(10)`
- `Invoke ClassroomStoryTest::theClassroomCondition(full)`
- if *`assertEquals`* *fails and throws* `ComparisonFailure` restore object `TEST`
- if any Then sentence has failed throw `StoryTestException`

או אם ניקח את הסיפור:

Given a classroom that the number of seats in it is 1337
When the the number of students in the classroom is 1378 and the number among them that are standing is 42 (זה משפט אחד)
Then the classroom is not-full

ונריץ את הסיפור באמצעות המתודה **testOnNestedClasses** עם מחלקת הטסט `ClassroomStoryDerivedTest`, סדר הפעולות שנעשה יהיה כך:

- Create object `TEST` of class `ClassroomStoryDerivedTest::InnerClass`
- `Invoke ClassroomStoryDerivedTest::InnerClass::aClassroom(1337)`
- BackUp object `TEST`
- `Invoke ClassroomStoryTest::InnerClass::classroomIsWIthStandingStudents(1378, 42)`
- `Invoke ClassroomStoryTest::theClassroomCondition(not-full)`
- if *`assertEquals`* *fails and throws* `ComparisonFailure` restore object `TEST`
- if any Then sentence has failed throw `StoryTestException`

אם נניח לצורך הדוגמא שקיימת גם המתודה classroomNoiseCondition במחלקה Classroom
וגם נניח שהמתודה theClassroomCondition השתנתה ל:

```
@Then("the classroom is &condition and &noiseCondition")
 public void theClassroomCondition(String condition, String noiseCondition)
{
       Assert.assertEquals(condition, classroom.classroomCondition());
       Assert.assertEquals(noiseCondition,
classroom.classroomNoiseCondition());
   }
```

עבור הסיפור הבא:

Given a classroom that the number of seats in it is 1337
When the the number of students in the classroom is 1378 and the number among them that are standing is 42 (זה משפט אחד)
Then the classroom is not-full and quiet or the classroom is full and noisy

נריץ את הסיפור באמצעות המתודה **testOnNestedClasses** עם מחלקת הטסט
`ClassroomStoryDerivedTest`, סדר הפעולות שנעשה יהיה כך:

- Create object `TEST` of class `ClassroomStoryDerivedTest::InnerClass`
- `Invoke ClassroomStoryDerivedTest::InnerClass::aClassroom(1337)`
- BackUp object `TEST`
- `Invoke ClassroomStoryTest::InnerClass::classroomIsWIthStandingStudents(1378, 42)`
- `Invoke ClassroomStoryTest::theClassroomCondition(not-full, quit)`
- if *`assertEquals`* *fails and throws* `ComparisonFailure`:
    - `WE SHOULD TRY THE SECOND SET OF PARAMS: Invoke ClassroomStoryTest::theClassroomCondition(full, noisy)`
    - if *`assertEquals`* *fails and throws* `ComparisonFailure`:
        - restore object `TEST`
- if any Then sentence has failed throw `StoryTestException`

עכשיו, לאחר כתיבת הרצת כל הסיפורים, ניתן לממש את המחלקה Classroom כך שכל הסיפורים יורצו בהצלחה:

```
public class Classroom {
   private Integer freeSpace;
   public Classroom(Integer capacity) {
       this.freeSpace = capacity;
   }

   public void addStudents(Integer numberOfStudents){
       this.freeSpace -= numberOfStudents;
   }

   public void addBrokenChairs(Integer numberOfBrokenChairs){
       this.freeSpace -= numberOfBrokenChairs;
   }
   public String classroomCondition(){
       return this.freeSpace > 0 ? "not-full" : "full";
   }
}
```

למעשה ניתן לראות שכך הצלחנו ליצור מערכת שיודעת להריץ סיפורי BDD ולבדוק אותם. כל מה שצריך לספק לה זה את מחלקת הבדיקות עם מתודות מתויגות.

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.
- לשם הבהרה: הטסטים שלנו יכללו מחלקות (כגון Classroom), מחלקות בדיקות (כגון ClassroomStoryTest), ואנחנו נבדוק את נכונות הריצה של 2 הפונקציות testOnInheritanceTree, testOnNestedClasses במחלקה StoryTesterImpl אשר אתם מממשים.

אין להגיש את הטסטים שלכם!

הוראות הגשה

- הגשת החלק הרטוב תתבצע אלקטרונית בלבד, יש לשמור את אישור השליחה! יש להגיש קובץ בשם OOP3_<ID1>_<ID2>.zip המכיל:

```
OOP3_<ID1>_<ID2>.zip
    |_ readme.txt
    |_ Given.java
    |_ When.java
    |_ Then.java
    |_ StoryTesterImpl.java
    |_ StoryTestExceptionImpl.java
    |_ <your additional files, if needed, no tests!>
```

- הגשה שלא לפי ההוראות תגרור הורדת ציון בהתאם.

בהצלחה!