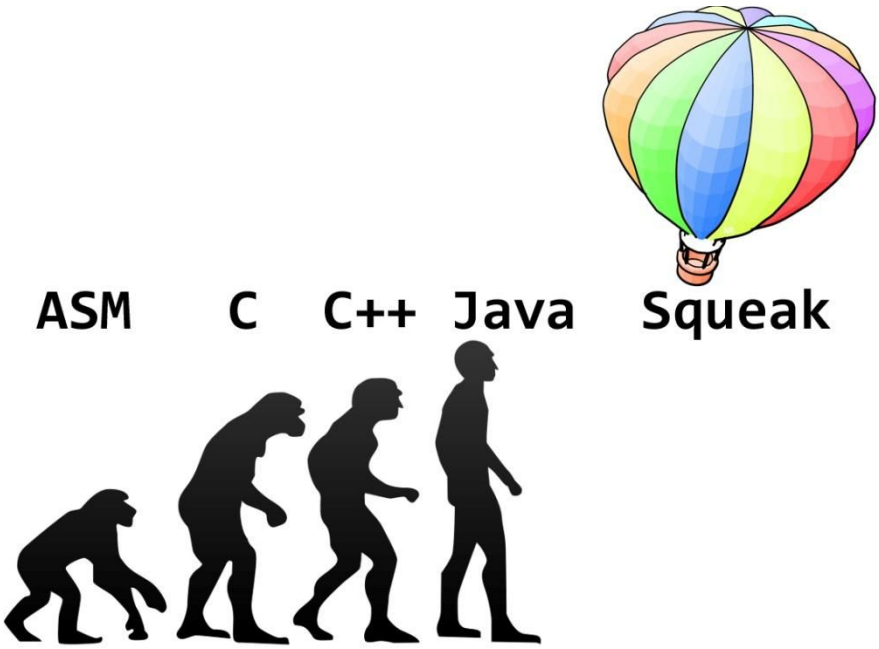
236703 – תכנות מונחה עצמים, חורף 2020

תרגיל 3: Squeak מתקדם



מבוא

1. בתרגיל זה נלמד את שפת Squeak לעומק תוך שימוש בחלק מהיכולות של השפה הכוללים יצור מחלקות ויירוט הודעות הנשלחות לאובייקט. בעזרת כלים אלו נוסיף ממשקים (Interfaces), מתודות ברירת-מחדל (Default Methods) ובדיקות טיפוסים בעת הפעלת מתודה לשפה.
2. על המחלקות להשתייך לקטגוריה חדשה בשם "OOP3".
3. בכדי להימנע מטעויות, אנא עיינו ברשימת ה FAQ המתפרסמת באתר באופן שוטף.
4. אחראי על התרגיל: **נתן בגרוב**. שאלות יש לשלוח ל natanb@cs עם הנושא: "HW3 236703"
5. מועד הגשה: **24/12/19 בשעה: 23:55**
6. **<u>יש לממש את תרגיל בית זה בסביבת Squeak 5.1</u>**.

שימו לב!

- **מכיוון שאנחנו לא רוצים לשנות את המנגנונים המובנים של Squeak, נממש מחלקה `MyObject` שתתמוך בשינויים של תרגיל בית זה.**

- **כל החריגות שיזרקו בתרגיל זה יהיו מסוג `AssertionFailure` - בסוף התרגיל תמצאו הסבר קצר.**

- **התרגיל מחולק ל- 3 חלקים, וכל חלק מסודר באופן הבא:**
  - מבוא, תיאור והגדרות
  - TO DO List ☑ עם מספור של המתודות עבור החלק
  - הערות הנחות עם מספור של המתודות עבור החלק והערות כלליות
  - מקרים לא חוקיים ותיאור חריגות עם מספור של המתודות עבור החלק

- שימו לב ל- **הנחות, מקרי קצה והערות לכלל התרגיל** – מופיע בסוף התרגיל.

חלק א' – אילוצים על טיפוסים[1]

מבוא, תיאור והגדרות:

נרצה להוסיף ל- Squeak מנגנון אשר יבצע **בדיקת טיפוסים** על הארגומנטים המועברים למתודות שיהודרו באופן בו המנגנון תומך.

מכיוון ש- Squeak שפה דינמית, אנו לא יכולים לזרוק חריגות בזמן ההידור של המתודה, מכיוון שאנו לא יודעים אילו פרמטרים יועברו בפועל, ולכן אנו נזרוק חריגה **בזמן ריצת המתודה** (מתוך המתודה).

הגדרה (1):

מתודה ללא מימוש / מתודה ריקה היא:

- מתודה שפרט לחתימתה (Selector וארגומנטים), לא רשום דבר. **שימו לב, שכפי שהוזכר בתרגול, מתודות ב- Squeak מחזירות את `self` כברירת מחדל, ולכן הגדרה זו נחוצה.**

TO DO:

יש להוסיף ל-`MyObject` את המתודות הבאות:

☑ **מתודה (1):** יש לממש את מתודת **המחלקה** הבאה:

[1] A.k.a. **Type Constrains**, you will meet this subject in the end of the semester. You don't need any former knowledge of it in this assignment.

compile: aSourceCode **where:** anOrderedCollection

- תפקיד מתודה זו הוא להדר את הקוד aSourceCode (להוסיף מתודת מופע שהחתימה והמימוש שלה מופיעות ב- aSourceCode למחלקה שלה שלחו את ההודעה) תחת האילוצים המופיעים ב- anOrderedCollection.
- לדוגמא:

```
A compile:
'foo: a bar:b baz: c
| var1 |
       var1 := a + c.
" just a comment, nothing special "
       ^ (var1 * var1)'
where: #(Integer nil Number).
```

הקוד הבא יוסיף את המתודה foo: bar: baz: למחלקה A, '**ויזריק**' לתוך הקוד של המתודה בדיקות על הטיפוסים של הארגומנטים: **a** יכול להיות מטיפוס Integer, **b** יכול להיות מכל טיפוס, **c** יכול להיות מטיפוס Number.

<u>הערות והנחות:</u>

עבור מתודה (1) compile: where:

- האילוצים מגדירים יחס של **isKindOf** ולא יחס מדוייק (isMemberOf), כלומר, הארגומנט c יכול להיות, למשל, Integer.
- אם אין אילוצים על הטיפוס, יועבר nil באינדקס המתאים.
- אם אין ארגומנטים למתודה, יועבר anOrderedCollection ריק.
- אין צורך לבצע בדיקה על הטיפוסים של aSourceCode ו- anOrderedCollection.

שימו לב לפורמט הנדרש.

<u>מקרים לא חוקיים ותיאור חריגות:</u>

עבור מתודה (1) compile: where:

- מספר האיברים ב- anOrderedCollection שונה ממספר הארגומנטים שהמתודה מקבלת – **חריגה 1.1. החריגה תיזרק מהמתודה המהדרת!**

```
'Can not compile method, number of arguments is not equal to the number of constraints!'
```

- אחד או יותר מהטיפוסים שהועברו להודעה לא תואם את האילוץ שאיתו היא מהודרת. למשל, ב- a מועברת מחרוזת – **חריגה 1.2. שימו לב שהחריגה אמורה להיזרק מהמתודה המהוַדרת, 'בזמן ריצה'!**

```
'Type mismatch! Argument #NUMBER should be #CLASS'
```

כאשר #NUMBER זה האינדקס של הארגומנט (החל מ- 1, כפי שכבר למדתם) ו- #CLASS זה האילוץ.

במקרה ויש יותר מאי-תאימות אחת, יש להחזיר את אי-התאימות של הארגומנט הראשון מבין הארגומנטים הבעיתיים.

לדוגמא:

```
'Type mismatch! Argument 2 should be SomeClass'
```

חלק ב' – Interfaces (ממשקים) ו- Default Methods (מתודות ברירת-מחדל)

מבוא, תיאור והגדרות:

בהמשך הקורס, נלמד לעומק על הורשה מרובה ב ++C ועל הקשיים בשימוש בה. כפי שלמדנו בתרגול 3, Java (וגם #C) החליטה על חלופה להורשה מרובה ע"י שימוש בממשקים (interfaces).
ב- Java קיימת הגבלה של ירושה ממחלקה אחת בלבד, אך תמיכה **במימוש** (Implements) **מספר כלשהו של ממשקים**, וכך, מחלקה יכולה להתחייב לחוזה מול מס' רב של ממשקים ובכך להעשיר את הפרוטוקול שלה כלפי חוץ.

בתרגיל זה, אנו נוסיף גרסה משלנו ל Interfaces ב- Squeak, תחת הגבלות מסוימות שיפורטו בהמשך. בנוסף, הממשקים שלנו יתמכו במתודות ברירת מחדל, שקיימות בממשקים ב- Java 8 כפי שראיתם בתרגול, תחת הגבלות מסוימות שיפורטו בהמשך.

מכיוון ש- Squeak אינה תומכת בממשקים כחלק מהשפה, נשתמש במחלקה הרגילה של Squeak במקום, ו"נלביש" עליה את הרעיון של ממשק:

- כל מחלקה (או ממשק) היורשת מ-MyObject תכיל אוסף של הממשקים שהיא מתנהגת כמוהם (הגדרה בהמשך)
- ממשק יהיה חייב לרשת **ישירות** מ- MyObject
- כל מחלקה (או ממשק) תכיל דגל isInterface שתפקידו כשמו כן הוא

מרגע זה והילך (אלא אם יצוין בפירוש אחרת), הכוונה ב*ממשק*, הוא לכזה שממומש לפי ההגדרות בתרגיל שלהלן.

הגדרה (1):

מתודה היא ***מתודת ברירת מחדל*** (Default Method) אם:

- שם המתודה מתחיל במחרוזת **default**. כלומר, defaultBar היא מתודת ברירת מחדל.
- כאשר המתודה מוגדרת במחלקה שאינה ממשק, מתודה זו היא מתודה רגילה לכל דבר.

1. בתרגיל זה נניח שהמילה default היא חלק **משם המתודה**. כלומר, Bar ו- defaultBar הן 2 מתודות שונות (אחת לא דורסת את השניה ואין ביניהן שום קשר).
2. מתודות ברירת מחדל יהודרו אך ורק באמצעות compile: (ולא, למשל compile: where :)
3. בתרגיל זה נניח שמתודת ברירת מחדל יכולה לקבל 0 או 1 ארגומנטים מטיפוס Integer, והמתודה לא ניגשת לאף שדה או מתודה אחרת. דוגמאות למתודות ברירת מחדל תקינות:

```
defaultMultByTwo: x
^ (2*x)
```

```
defaultRandom
^ 42
```

הגדרה (2):

ממשק הוא (בתור מוסכמה, שמו יתחיל באות I):

- **ממשק** בסקוויק (`interface`) הוא מחלקה שאינה מכילה שדות, ויכולה להכיל מתודות.
- **ממשק** יכול להכיל מתודות ללא מימוש (לפי הגדרה (1) בחלק הקודם)
- **ממשק** לא יכיל מתודות עם מימוש, אלא אם הן מתודות ברירת-מחדל (לפי הגדרה (1) בחלק הנוכחי).
- **ממשק** יכיל `true` בדגל `isInterface`.

הגדרה (3):

מחלקה (או ממשק) X ***מתנהגת כמו ממשק*** I אם מתקיים אחד (או יותר) מהבאים:

- **הממשק** I מופיע ב- behavesLike של אותה **מחלקה (או ממשק)** X
- מחלקת האב של **המחלקה** X מתנהגת כמו הממשק I (שימו לב לרקורסיה שאמורה להמשיך עד `MyObject`, לא כולל `MyObject`)
- אחד (או יותר) מהממשקים ש**המחלקה (או ממשק)** X מתנהגת כמוהו מתנהג כמו הממשק I (שימו לב לרקורסיה שאמורה להמשיך עד `MyObject`, לא כולל `MyObject`)
- ממשק I מתנהג כמו הממשק עצמו (I), גם אם אינו מופיע ב- behavesLike

הגדרה (4):

מתודה **יוצרת התנגשות** (***ambiguous***) עבור מחלקה (או ממשק) X אם מתקיים:

- המתודה קיימת ב- 2 או יותר **ממשקים** ש- X מתנהג/ת כמוהם.

הגדרה (5):

מחלקה X ***מממשת ממשק*** I אם מתקיים:

- X מתנהגת כמו ממשק I
- כל המתודות ש- I מכירה ממומשות ב- X, **פרט למתודות ברירת מחדל** (אותן ניתן לדרוס וניתן שלא)
  - כלומר, X מכירה את כל המתודות שהממשק מכיר (ההגדרה רקורסיבית: ממשק מכיר מתודה אם הגדיר אותה או אם היא מוגדרת באחד מהממשקים שהוא מתנהג כמוהם), פרט, אולי, למתודות ברירת מחדל

- שימו לב שזה לא אומר שכל המתודות חייבות להיות מוגדרות במחלקה הנוכחית, חלק מהמתודות יכולות להיות ממומשות במחלקת האב למשל.
- שימו לב שההגדרה היא רקורסיבית - לדוג' X מתנהגת כמו IB ו- IB מתנהג כמו הממשק IA - המחלקה X נדרשת לממש (לדרוס) הן את המתודות ב- IB וגם את המתודות ב- IA
- שימו לב לאופן הריק: אם הממשק לא מכיר מתודות, הוא נחשב כממומש אמ"מ קיימת מחלקה אשר מתנהגת כמוהו.

הגדרה (6):

ממשק I ***ממומש*** אם מתקיים:

- קיימת מחלקה X הממשת את I.

TO DO:

יש להוסיף ל-`MyObject` את המתודות הבאות:

☑ יש להוסיף ל- `MyObject` את ה- **`class instance variables`**[2] הבאים:

שדה (1): **`behavesLike`** – סט (או `collection` אחר לבחירתכם) של **ממשקים** שהמחלקה/ממשק מתנהגים כמוהם.

שדה (2): **`isInterface`** – שדה בוליאני שיכיל `true` אם מדובר בממשק ו- `false` אם מדובר במחלקה שאינה ממשק.

- **אין להוסיף למחלקה זו שדות נוספים! (וגם אין צורך)**

יש להוסיף ל-`MyObject` את מתודות המחלקה הבאות:

☑ **מתודה (1):** יש לממש את מתודת המחלקה הבאה:

```
subclass: aSubclassName isInterface: isInterface behavesLike:
aCollection instanceVariableNames: instVarNames classVariableNames:
classVarNames poolDictionaries: poolDictionaries      category:
aCategoryName
```

- מתודה זו יוצרת מחלקת בן חדשה למחלקה הנוכחית.
- מתודה זו זהה למתודת יצירת subclass של השפה פרט להוספת 2 פרמטרים:
  - **`isInterface:`** `isInterface` ב- `isInterface` יועבר ערך Boolean שמהווה אינדיקציה האם אנו מגדירים ממשק או מחלקה 'רגילה' שאינה ממשק.

---

[2] מומלץ לקרוא על ההבדלים בין class variables, class instance variables, instance variables

- `behavesLike: aCollection`. ב- `aCollection` יתקבלו שמות **הממשקים** שהממשק/מחלקה מתנהגים כמוהם, לדוגמא: `{IA . IB . IC}`, או ריק `{ }` אם אין כאלה.

- על המתודה לדאוג ליצירת המחלקה בעץ הירושה הרגיל של Squeak ולאתחל את השדות של המנגנון החדש. (**רמז:** לצורך אתחול, חפשו את המתודה `instVarNamed:put`:)
- **על המתודה להחזיר את <u>אובייקט המחלקה</u> החדשה שנוצרה.**

☑ **מתודה (2):** יש **לדרוס** את מתודת המחלקה הבאה:

```
subclass: aSubclassName instanceVariableNames: instVarNames
classVariableNames: classVarNames poolDictionaries: poolDictionaries
category: aCategoryName
```

- אנו נרצה לתמוך גם באופן יצירת ה- subclass הרגיל, כלומר, קריאה למתודה זו תיצור לנו מחלקה 'רגילה' שאינה ממשק, ולא מתנהגת **ישירות** כמו אף ממשק. המחלקה תתנהג כמו כל הממשקים שמחלקת האב שלה מתנהגת כמוהם. זה רלוונטי כמובן אם יורשים ממחלקה כלשהי שיורשת מ- `MyObject` או מ- `MyObject` עצמו.

☑ **מתודה (3):** יש לממש את מתודת המחלקה הבאה: `isInterface`

- המתודה תחזיר ערך Boolean האם המחלקה היא **ממשק** או לא.

☑ **מתודה (4):** יש לממש את מתודת המחלקה הבאה: `behavesLike`

- המתודה תחזיר **Set** של כל **הממשקים** שהממשק/מחלקה מתנהג/ת כמוהם (לפי הגדרה 3 מעלה).

☑ **מתודה (5):** יש לממש את מתודת המחלקה הבאה: `isImplemented`

- המתודה תחזיר `true` אם הממשק **ממומש**, ו- `false` אחרת (התייחסו ביתר תשומת לב למתודות ברירת-מחדל)

☑ **מתודה (6):** יש לממש את מתודת המחלקה הבאה: `ambiguities`

- המתודה תחזיר SortedCollection של כל המתודות **שיוצרות התנגשות**.

ממויינות בסדר עולה לפי שם.

- לדוגמא, אם יש למחלקה A ארבע מתודות שיוצרות התנגשות –
`#foo:baz:, #a, #foo:, foo#`, נקבל כפלט (שימו לב לסדר)

```
Transcript show: (A ambiguities)
 a SortedCollection(#a #foo #foo: #foo:baz:)
```

☑ **מתודה (7):** יש לדרוס מתודת מחלקה: `(You will have to find out what method to override)`

- על מנת לאכוף **ולמנוע יצירת מופעים של ממשקים**, תצטרכו לדרוס מתודה כלשהי וכאשר ינסו ליצור מופע של **ממשק**, המתודה תזרוק חריגה.

☑ **קטע קוד (8):** יש להוסיף קטע קוד למתודה: `(You will have to find out what method to change)`

- קטע קוד זה יאכוף מצב בו מנסים להדר מתודה עם מימוש, שאינה מתודת ברירת מחדל, בממשק. במקרה כזה, המתודה (המתודה שלה אתם מוסיפים את קטע הקוד – ולא המתודה שמוסיפים לממשק) תזרוק חריגה.

<u>הערות והנחות:</u>

עבור מתודה (1) `subclass: isInterface: behavesLike: instanceVariableNames: classVariableNames: poolDictionaries: category:`

- ניתן להניח שלא יועברו כפילויות ב- `aCollection`.
- יכולים להתקבל ממשקים שהמחלקה כבר מתנהגת כמוהם, לדוגמא אם המחלקה יורשת ממחלקה אב שמתנהגת כמוהם. זה מצב חוקי, אך ניתן להתעלם מהם, כי המחלקה המהודרת גם ככה תתנהג כמוהם.

עבור מתודה (4) `behavesLike`

- אם אין ממשקים שהמחלקה / ממשק מתנהגים כמוהם, יוחזר סט ריק.
- שימו לב שעל פי הגדרה (3), יש הבדל בין שליחת ההודעה לממשק או למחלקה שאינה ממשק.

עבור מתודה (6) `ambiguities`

- אם אין התנגשויות, יוחזר SortedCollection ריק.
- כמו שראיתם, מתודות מתווספות ע"י `compile:` או `compile:where` מהחלק הקודם, ולכן יתכן שתופענה מתודות חדשות בין 2 קריאות למתודה זו, אם נוספו כאלה בין הקריאות.

עבור קטע קוד (8)

- שימו לב לאופן בו יהודרו מתודות. הפורמט מופיע בסוף המסמך, תחת **הנחות, מקרי קצה והערות לכלל התרגיל**.

<u>מקרים לא חוקיים ותיאור חריגות:</u>

עבור מתודה (1) `subclass: isInterface: behavesLike: instanceVariableNames: classVariableNames: poolDictionaries: category:`

- יצירת **ממשק** שלא יורש **ישירות** מ- MyObject – **חריגה 2.1.**

```
'Interfaces must derive from MyObject!'
```

- יצירת **ממשק** עם State – **חריגה 2.2.**

```
'Interfaces can not have state!'
```

- יצירת **מחלקה שאינה ממשק <u>שיורשת</u> מממשק** – **חריגה 2.3.**

```
'Classes can not derive from an interface!'
```

- מתן התנהגות (behavesLike) של **מחלקות שאינן ממשקים**, כלומר, האוסף לא מכיל ממשקים בלבד – **חריגה 2.4.**

```
'Can not behave like a non-interface!'
```

שימו לב שיש חשיבות לסדר החריגות! אם קרו מספר מקרים לא חוקיים, יש לזרוק את החריגה הנמוכה ביותר! למשל, עבור ממשק עם State שאינו יורש מ- MyObject (חריגות 2.1+2.2), תיזרק חריגה 2.1.

עבור מתודה (5) `isImplemented`

- אם המתודה מופעלת לא על ממשק – חריגה 2.5.

```
'#CLASS is not an interface!'
```

כאשר #CLASS זו המחלקה לה שלחו את ההודעה.

עבור מתודה (7) `(You will have to find out what method to override)`

- אם המתודה מופעלת על ממשק – חריגה 2.6.

```
'Interfaces can not be instantiated!'
```

עבור קטע קוד (8)

- אם מנסים להדר מתודה עם מימוש בממשק – חריגה 2.7.

```
'Interfaces are not allowed to have methods that define behavior!'
```

חלק ג' – מחלקות אבסטרקטיות, isKindOf

<u>מבוא ותיאור:</u>

<u>(מחלקה אבסטקרטית):</u>

מכיוון שמחלקות **לא יורשות מממשקים** בדרך הירושה של Squeak, מובן שכאשר נשלח הודעה (נפעיל מתודה) ש**הממשק** מכיר (מתודה המוגדרת אצל הממשק), למופע של **המחלקה**, תיזרק חריגה doesNotUnderstand.

הגדרה (1):

מחלקה היא ***אבסטרקטית*** אם מתקיים :

- קיים ממשק שהמחלקה מתנהגת כמוהו ולא מממשת אותו.
  - שימו לב שמחלקה לא תיחשב אבסטרקטית אם המתודות היחידות שהיא לא מכירה הן מתודות ברירת מחדל של הממשקים שהיא מתנהגת כמוהם

כפי שוודאי הבנתם מהשם, אנו לא נאפשר ליצור מופעים של המחלקה הזו.

<u>(isKindOf):</u>

לסיום, נרצה לבצע אינטגרציה של חלקים א' ו- ב', על מנת לתמוך ב- **ממשקים בתור Type Constraints**, כלומר, שגם ממשקים יוכלו להיות אילוצים.

<u>TO DO:</u>

יש להוסיף ל-`MyObject` את המתודות הבאות:

☑ **קטע קוד (1):** יש להוסיף קטע קוד למתודה: `(You will have to find out what method to change)`

- קטע קוד זה יאכוף מצב בו מנסים ליצור מופע של מחלקה אבסטרקטית. במקרה כזה, המתודה תזרוק חריגה.

☑ **מתודה (2):** יש לממש את מתודת **המופע** הבאה:

```
isKindOf: aClassOrInterface
```

- המתודה תחזיר ערך Boolean שמציין אם האובייקט שהמתודה הופעלה עליו עמד בתנאים של `isKindOf` המקורית או מתנהג כמו `aClassOrInterface` . אם האובייקט לא עומד ב- 2 התנאים הללו, יש להחזיר `false`.

☑ **קטע קוד (3):** יש להוסיף קטע קוד למתודה: `(You will have to find out what method to change)`

- מטרת קטע קוד זה היא לגרום למחלקה **להכיר את כל מתודות ברירת המחדל של הממשקים שהיא מתנהגת כמוהם**. זאת על מנת שבעת יצירת מופע של המחלקה הזו, המופע יוכל לקבל את ההודעות הללו.

<u>הערות והנחות:</u>

עבור קטע קוד (1)

- שימו לב:
  - עדיין ניתן לרשת ממחלקה אבסטרקטית; תיתכן מחלקה אבסטרקטית שיורשת ממחלקה שאינה כזו, ותיתכן מחלקה שאינה אבסטרקטית שיורשת ממחלקה אבסטרקטית.
  - מחלקה אבסטרקטית יכולה להפוך למחלקה ממשית, על ידי מימוש כל המתודות המופיעות בממשקים שהיא מתנהגת כמוהם אך איננה מכירה (שימו לב שההגדרה היא רקורסיבית - לדוג' אם המחלקה B מתנהגת כמו IB והממשק IB מתנהג כמו הממשק IA - המחלקה B נדרשת לממש הן את המתודות ב- IB <u>וגם</u> את המתודות ב- IA, פרט כמובן למתודות ברירת-מחדל).
  - מחלקה ממשית יכולה להפוך לאבסטרקטית, על ידי הוספת מתודה חדשה באחד מהממשקים שהמחלקה מתנהגת כמוהם.
  - **על מנת לא לגרום להתנהגות לא חוקית, ניתן להניח שמחלקה לא תהפוך לאבסטרקטית אם יש מופע שלה.**
  - **כמו כן, ניתן להניח שלא יתווספו מתודות ברירת מחדל לממשק שהמחלקה מתנהגת כמוהו.**

עבור קטע קוד (3)

- ניתן להניח שלא יתווספו מתודות ברירת מחדל לממשק שהמחלקה מתנהגת כמוהו. כלומר, לאחר **יצירת המחלקה** (אובייקט המחלקה – לא מופע) – לא יתווספו מתודות ברירת מחדל לממשקים הללו (אך יתכן שיתווספו מתודות ללא מימוש שאינן מתודות ברירת מחדל כפי שנכתב בהערה הקודמת).
  - ניתן להניח שלא יבדקו מקרים בהם יש התנגשויות של מתודות ברירת-מחדל (עם זאת, יכולות להיות כאלה).

מקרים לא חוקיים ותיאור חריגות:

עבור קטע קוד (1)

- אם מנסים ליצור מופע של מחלקה אבסטקרטית – חריגה 3.1.

```
'Can not instantiate an Abstract class!'
```

סיווג חריגות

חריגות בתרגיל ידווחו על ידי שימוש ב-

`AssertionFailure signal: 'ERROR'`

שזורק חריגה מסוג `AssertionFailure` עם טקסט ERROR.

```
1.1: 'Can not compile method, number of arguments is not equal to the number of constraints!'
1.2: 'Type mismatch! Argument #NUMBER should be #CLASS'
2.1: 'Interfaces must derive from MyObject!'
2.2: 'Interfaces can not have state!'
2.3: 'Classes can not derive from an interface!'
2.4: 'Can not behave like a non-interface!'
2.5: '#CLASS is not an interface!'
2.6: 'Interfaces can not be instantiated!'
2.7: 'Interfaces are not allowed to have methods that define behavior!'
3.1: 'Can not instantiate an Abstract class!'
```

- בתרגיל זה אנו עובדים רק עם חריגות מסוג **`AssertionFailure`**.
- שימו לב ל- #CLASS ו- #NUMBER, יש להחליף אותם בשם המחלקה ומספר הארגומנט בהתאם.
- לנוחיותכם, יסופק לכם קובץ txt לתרגיל עם הפירוט של החריגות שלמעלה.

הנחות, מקרי קצה והערות לכלל התרגיל

- ניתן להניח שלפני הרצה הטסטים, תורצנה השורות הבאות:

```
MyObject instVarNamed: 'isInterface' put: false.
MyObject instVarNamed: 'behavesLike' put: {}.
```

- ניתן להניח כי ה- class instance variables שהוספתם בחלק ב' (behavesLike, isInterface) לא ישונו לאחר הוספת המחלקה.
- לא יהודרו מתודות עם אילוצים (constraints) בממשקים, **כלומר, לא תישלח ההודעה `:compile:where`** לממשק.
- לא יהודרו מתודות זהות (בעלות אותו selector) פעמיים בממשקים.
- לבדיקת האם המתודה מכילה מימוש או לא - מתודות לא יכילו שורה ריקה, או הערה בלבד. ניתן להתעלם ממקרים אלה, שכן הם לא יבדקו. כאן לא מבוא למדמ"ח.

- **כל המתודות שיבדקו בטסטים של כל החלקים יהודרו כמו בדוגמא בחלק א', באופן הבא:**
  - **השורה הראשונה** תכיל את ה- selector והארגומנטים בלבד, כאשר בין ה- selector לבין הנקודותיים לא יהיה רווח, ובין הנקודותיים לבין ה- arguments יכול להיות רווח ויכול לא להיות רווח.
  - **<u>אם קיימים</u> temporary variables**, **השורה השניה** תכיל את ה- temporary variable names של המתודה ואותם בלבד, כאשר יש רווח בין | לבין המשתנים, גם בהתחלה וגם בסוף.
  - **שאר הקוד** יכיל את המימוש של המתודה וגם הערות במידת הצורך.

טיפים שימושיים והנחיות

- יתכן שיהיה לכם יותר קל לפתור את התרגיל אם תתייחסו לחיפוש המתודות במעלה הירושה בתור חיפוש בגרף.
- לפני שאתם ניגשים לפיתרון, מומלץ לעבור שוב על התרגול המתקדם ובפרט על התרשים המציג את מודל 5 הרמות בשפה.
- ניתן להגדיר **מתודת מחלקה** (בניגוד למתודת מופע) ע"י לחיצה על כפתור ה- class, הממוקם מתחת לחלון המחלקות ב- ObjectBrowser. שם מגדירים גם class-instance variables.
- אחת ממטרות התרגיל היא לאפשר לכם לחקור את יכולות השפה. לכן, חלק עיקרי מהפתרון הוא חיפוש אחר מתודות ומחלקות שונות אשר ישמשו אתכם לצורכי התרגיל. כדאי להיעזר בתיבת החיפוש של Squeak או לנסות לחפש מחלקות ע"פ הקטגוריות השונות ב-Object Browser. מומלץ להתחיל ב-Behavior ולחפש בו מתודות לפי הקטגוריות. הצפי הוא שתבינו בעצמכם איך לבצע את הפעולות הנדרשות בSqueak, ולכן שאלות מהסגנון "איך אני עושה כך וכך ב-Squeak" לא יענו אלא אם אתם מראים שכבר ניסיתם ולא הצלחתם למצוא איך לבצע את מה שאתם מנסים.
- אין להוסיף מחלקות נוספות מעבר לאלו שנתבקשתם לממש בתרגיל.
- אין לשנות מחלקות נוספות מעבר לאלו שהתבקשתם לשנות במפורש בתרגיל.
- אין לדרוס או לשנות מתודות שהשם שלהן מתחיל ב-basic.
- מותר להוסיף מתודות עזר כרצונכם, אסור להוסיף שדות נוספים.
- אין להדפיס דבר לפלט (`Transcript`). אם אתם מדפיסים לצורך בדיקות, הקפידו להסיר את ההדפסות לפני ההגשה.

- **יש לתעד כל קטע קוד שאינו טריוויאלי. יש לתעד בקצרה כל מתודת עזר שהגדרתם. בכל אופן, אין צורך להפריז בתיעוד.**
- אם אתם מרגישים שנתקעתם – Google is your friend. אם אתם מתקשים למצוא תוצאות שימושיות עבור Squeak, נסו לחפש את אותן יכולות ב-Smalltalk (שכן המידע הקיים עבורה הוא נרחב יותר ו- Squeak היא למעשה ניב שלה).

הוראות הגשה

- **בקשות לדחייה, מכל סיבה שהיא, יש לשלוח למתרגל האחראי על הקורס (נתן) במייל בלבד תחת הכותרת HW3 236703. שימו לב שבקורס יש מדיניות איחורים, כלומר ניתן להגיש באיחור גם בלי אישור דחייה – פרטים באתר הקורס תחת General info.**
- הגשת התרגיל תתבצע אלקטרונית בלבד (יש לשמור את אישור השליחה!)
- יש להגיש קובץ בשם `OOP3_<ID1>_<ID2>.zip` המכיל:
  - קובץ בשם `readme.txt` המכיל שם, מספר זיהוי וכתובת דואר אלקטרוני עבור כל אחד מהמגישים.
  - קובץ הקוד: `OOP3.st`. על הקובץ להכיל רק את מימוש המחלקה המוזכרת בתרגיל (`MyObject`) ומתודות לצורך פתרון התרגיל. **אין להגיש קוד נוסף, כמו טסטים**.
- נקודות יורדו למי שלא יעמוד בדרישות ההגשה (rar במקום zip, קבצים מיותרים נוספים, readme בעל שם לא נכון וכו')

בהצלחה!

(בהתחלה תהיו במצב הימני, אחר כך תרגישו כמו במצב השמאלי)