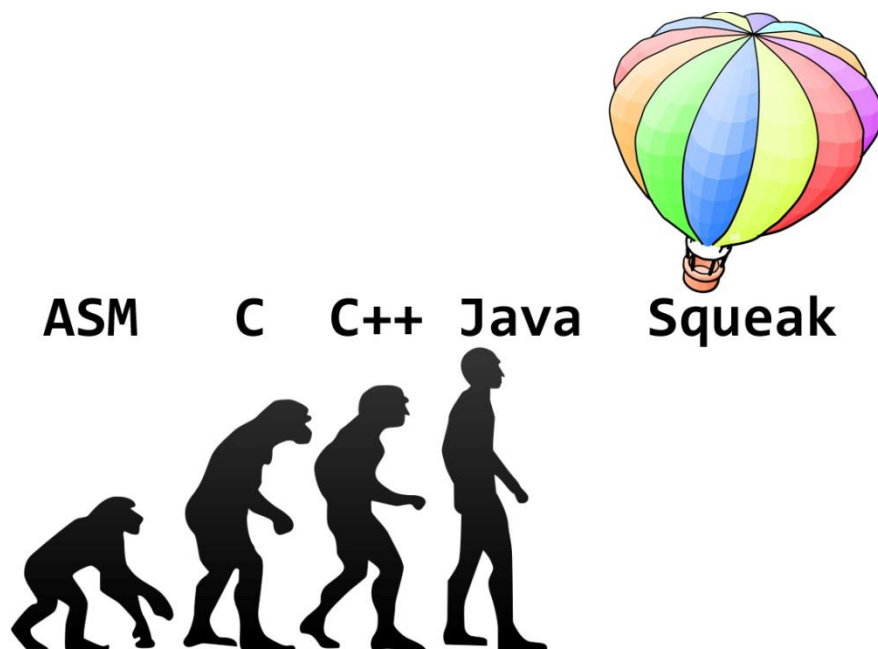


תרגיל 3: Squeak מתקדם



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)** 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: naseem yehya. שאלות יש לשלוח ל- naseem.yehya@campus.tecnion.ac.il עם הנושא: "236703 HW3".
5. מועד הגשה: 21/12/21 בשעה: 23:55
6. יש לממש את תרגיל בית זה בסביבת Squeak 5.1.

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 AssertionError - 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 טיפוסים¹

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, 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, 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:

`compile: aSourceCode where: anOrderedCollection`

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.

- תפקיד מתודה זו היא להדר את הקוד `aSourceCode` (להוסיף מתודת מופע שהחתימה והמימוש שלה מופיעים 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 + (2 * b).
"just a comment, nothing special"
^ (var1 * var1)'
where: #(Integer nil Number).
```

החלק המודגש הוא דוגמא לפורמט מתודה עבור קריאה ל- `compile:where:`. פרטים נוספים ב- [הנחות, מקרי קצה והערות לכלל התרגיל](#)

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

הערות והנחות:

compile: where: (1) עבור מתודה

- האילוצים מגדירים יחס של `isKindOf` ולא יחס מדויק (`isMemberOf`), כלומר, בדוגמא, 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`.

מקרים לא חוקיים ותיאור חריגות:

compile: where: (1) עבור מתודה

- מספר האיברים ב- `anOrderedCollection` שונה ממספר הארגומנטים שהמתודה מקבלת – חריגה 1.1, **שתיזרק מתוך המתודה: compile: where:**
- '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, בדוגמא, ב- `a` מועברת מחרוזת – חריגה 1.2, **שתיזרק מתוך המתודה שהידרתם.**

'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 הבעייתיים.

לדוגמא:

'Type mismatch! Argument 2 should be Integer'

חלק ב' – Interfaces (ממשקים)

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 ב**מימוש** (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 נוסף גרסה משלנו ל Interfaces ב-Squeak, תחת הגבלות מסוימות שיפורטו בהמשך.

מכיוון ש-Squeak אינה תומכת בממשקים כחלק מהשפה, נשתמש במחלקה הרגילה של Squeak, ונלביש עליה את הרעיון של ממשק:

- כל מחלקה (או ממשק) היורשת מ-MyObject תכיל אוסף של הממשקים שהיא מתנהגת כמוהם (שדה בשם behavesLike, 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 בממשק, הוא לכזה שממומש לפי ההגדרות בתרגיל:

הגדרה (2):

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

- **ממשק** בסקוויק (interface) הוא מחלקה שאינה מכילה שדות, ויכולה להכיל מתודות.
- **ממשק** לא יכיל מתודות עם מימוש. אך יכול להכיל מתודות ללא מימוש.
- **ממשק** יכיל true בדגל isInterface.

הגדרה (3):

מחלקה (או ממשק) X **מתנהגת כמו ממשק I** אם מתקיימים אחד (או יותר) מהבאים (בדומה למה שלמדנו על Java):

- **הממשק I** מופיע בשדה 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 שהמחלקה (או ממשק) X מתנהגת כמוהו מתנהג כמו הממשק I שימו לב לרקורסיה שאמורה להמשיך עד MyObject, לא כולל)
- 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 מתנהגת כמוהם.

הגדרה (5):

מחלקה X מממשת ממשק I אם מתקיימים כל הבאים (בדומה למה שלמדנו על Java):

- X מתנהגת כמו ממשק I
- כל המתודות של I ממומשות ב- X
 - כלומר, X מכירה את כל המתודות שרשומות בממשק
 - שימו לב שזה לא אומר שכל המתודות חייבות להיות מוגדרות במחלקה הנוכחית, חלק מהמתודות יכולות להיות ממומשות במחלקת האב.
 - שימו לב לאופן הריק (בדומה ל- `Clonable` ב-Java)

הגדרה (6):

ממשק I ממומש אם מתקיים:

- קיימת מחלקה X המממשת 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` – collection 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
```

² מומלץ לקרוא על ההבדלים בין `instance variables`, `class instance variables`, `class variables`

- מתודה זו יוצרת מחלקת בן חדשה למחלקה הנוכחית.
- מתודה זו זהה למתודות 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.
- - `behavesLike: aCollection` ב- `aCollection` יתקבלו שמות הממשקים שהממשק/מחלקה מתנהגים כמוהם, לדוגמא: {IC . IB . IA}, או ריק { } אם אין כאלה.
- על המתודה לדאוג ליצירת המחלקה בעץ הירושה הרגיל של Squeak ולא לחל את השדות של המנגנון החדש. (רמז: לצורך אתחול, חפשו את המתודה `instVarNamed:put:`)
- על המתודה להחזיר את אובייקט המחלקה 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. כלומר, על פי ההגדרה מעלה, המחלקה תתנהג כמו כל הממשקים שמחלקת האב שלה מתנהגת כמוהם.

□ **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 מתנהגת/ת כמוהם.

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

- המתודה תחזיר true אם הממשק **ממומש**, ו- false אחרת.

□ **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`, `#foo:baz:`, `#a`, `#foo`, נקבל כפלט (שימו לב לסדר)
 Transcript show: (A ambiguities)
 ← a SortedCollection(#a #foo #foo: #foo:baz:)

□ **מתודה (7):** יש לדרוס את מתודת המחלקה הבאה: (You will have to find out what method to override)

- על מנת לאכוף **ולמנוע יצירת מופעים** של ממשקים, תצטרכו לדרוס מתודה כלשהי (תחשבו איזו, ועדיף שתקראו את כל התרגיל לפני-כן), וכאשר ינסו ליצור מופע של **ממשק**, המתודה תזרוק חריגה.

□ **קטע קוד (8):** יש להוסיף קטע קוד למתודה שיתכן והוספתם / דרסתם לפני-כן.

- קטע קוד זה יאכוף מצב בו מנסים להדר מתודה עם מימוש בממשק. במקרה כזה, המתודה תזרוק חריגה.

הערות והנחות:

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 (5) `isImplemented`

- שימו לב שההגדרה היא הגדרה רקורסיבית.
- כמו שראיתם, מתודות מתווספות ע"י `compile:` או `compile:where:` מהחלק הקודם, ולכן יתכן שתופענה מתודות חדשות בין 2 קריאות למתודה זו, ומנשק שהיה ממומש יהפוך להיות ללא ממומש, ולהיפך.

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**.

מקרים לא חוקיים ותיאור חריגות:

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 שירשת מממשק – חריגה 2.3.
'Classes can not derive from an interface!'
- מתן התנהגות (behavesLike) של מחלקות שאינן ממשקים, כלומר, המערך 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!'

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!'
```

- 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.
- לא יהודרו מתודות זהות (בעלות אותו `selector`) פעמיים בממשקים.

כל המתודות שיבדקו בטסטים של כל החלקים יהודרו בפורמט הדוגמא מחלק א', כלומר:

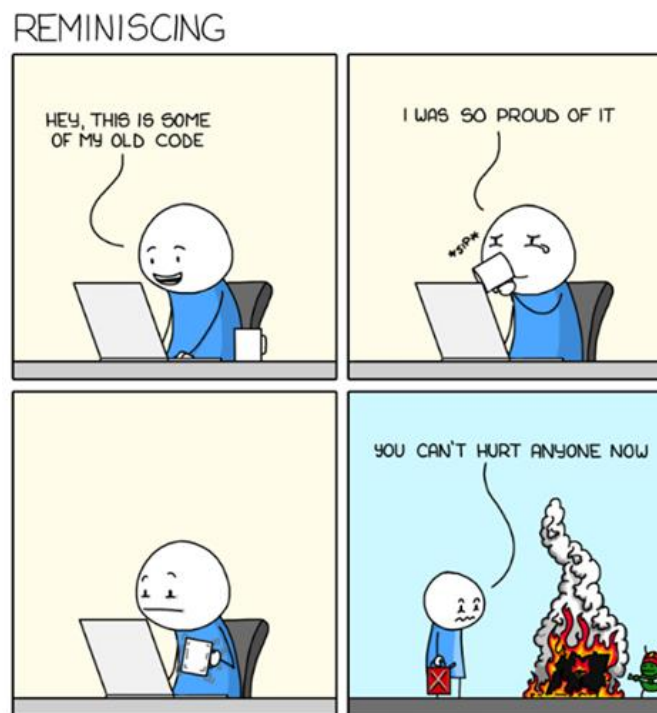
- **השורה הראשונה** תכיל את ה-`selector` והארגומנטים בלבד, כאשר בין ה-`selector` לבין הנקודותיים לא יהיה רווח, ובין הנקודותיים לבין ה-`arguments` יכול להיות רווח ויכול לא להיות רווח.
- **אם קיימים `temporary variables`**, **השורה השנייה** תכיל את ה-`temporary variable` `names` של המתודה ואותם בלבד, כאשר 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 למתודות 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 ב-`Squeak`, 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 (יבדק. איך reflection).
- 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 (גולאן) בלבד.
- 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 (למשל טסטים).
- נקודות יורדו למי שלא יעמוד בדרישות ההגשה (`rar` במקום `zip`, קבצים מיותרים נוספים, `readme` בעל שם לא נכון וכו')



...סטודנט שמסתכל על תרגיל בית 1 שלו אחרי שפתר את התרגיל הזה.

בהצלחה!