תרגיל 4: Squeak מתקדם

כללי

1. **מועד הגשה: 5/9/2019 בשעה 23:59.**
2. קראו היטב את ההוראות במסמך זה ובקוד שניתן לכם.
3. אחראי על התרגיל: **נתן**. שאלות על התרגיל יש לשלוח למייל natanb@cs עם הנושא: "236703 HW4".
4. הקפידו על קוד ברור, קריא ומתועד ברמה סבירה. עליכם לתעד כל חלק שאינו טריוויאלי בקוד שלכם. כמו כן הימנעו משכפול קוד והשתמשו במידת האפשר בקוד שמימשתם כבר.
5. מהירות ביצוע אינה נושא מרכזי בתרגילי הבית בקורס. בכל מקרה של התלבטות בין פשטות לבין ביצועים, העדיפו את המימוש הפשוט.
6. **<u>הגשה בזוגות בלבד.</u>**
7. **בכדי להימנע מטעויות, אנא עיינו ברשימת ה-FAQ המתפרסמת באתר באופן שוטף.**

מבוא

בתרגיל זה נלמד את שפת Squeak לעומק תוך שימוש בחלק מהיכולות של השפה הכוללים יצור מחלקות ויירוט הודעות הנשלחות לאובייקט. בעזרת כלים אלו נממש הורשה מרובה ו-Access Specifiers בSqueak.

Access Specifiers – מה זה?

ב- C++ נציין את נגישות ההורשה בעזרת אחד מ-3 ה-Access Specifiers הבאים: **public, private, protected**, הפועלים בצורה דומה ל-Access Specifiers של מתודות:
בהינתן מחלקות A ו-B כך ש-B יורש מ-A:

- **אם ההורשה היא private:** רק B "יודע" ש-B יורש מ-A, כלומר רק B יכול להפעיל מתודות של תת המחלקה A במופע B. שאר המחלקות לא יודעות על הירושה. בפועל, כל המתודות שהיו public/protected ב-A יהפכו ל-private ב-B.
- **אם ההורשה היא protected:** רק B והמחלקות שיורשות מ- B "יודעות" על ההורשה של B מ-A, כלומר רק מופעים של B או של מחלקות שיורשות מ-B יכולות להפעיל מתודות של תת המחלקה A. בפועל, כל המתודות שהיו public/protected יהפכו ל-protected ב-B ובמחלקות שיורשות מ-B.
- **אם ההורשה היא public:** כל המחלקות "יודעות" ש-B יורש מ-A, ולכן ניתן להפעיל מתודות של תת המחלקה A במופע של B גם מחוץ ל-B. נגישות המתודות שהיו ב-A תאכף כרגיל (כלומר private יהיה זמין רק ל-A, ו- protected יהיה זמין לכל מי שיורש מ-A ו-public יהיה זמין לכולם)

בתרגיל נרצה לממש תכונה דומה ב-Squeak. המימוש שלנו יהיה תקף רק על ההורשה המרובה שנממש בהמשך. (ולא על ההורשה הרגילה בשפה)

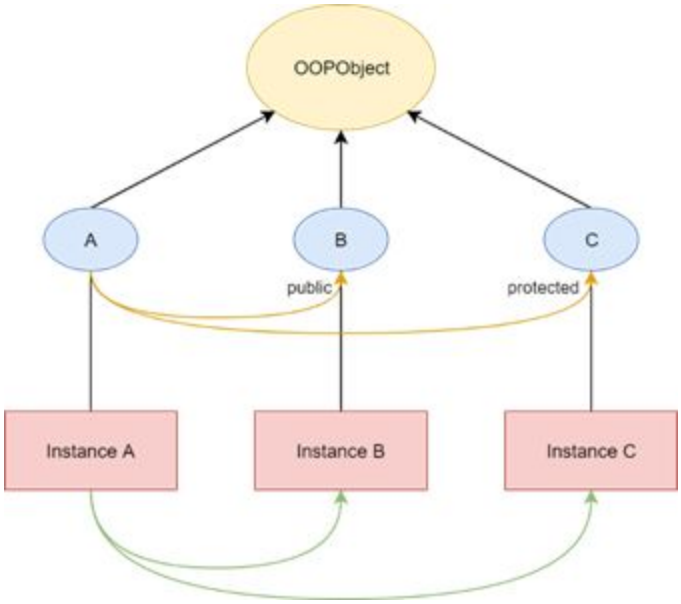
מנגנון ההורשה המרובה

מכיוון שאנחנו לא מעוניינים לשנות את מנגנון הירושה המובנה של Squeak, נממש את המחלקה **OOPObject** עם מבנה התומך במנגנון הירושה כמתואר בהמשך. כל המחלקות שנרצה שישתמשו ביכולות שנממש ירשו מ-OOPObject.

כל מופע של מחלקה היורשת מ-OOPObject יכיל אוסף של מופעים של המחלקות מהן היא יורשת (שימו לב: מדובר בהורשה המיוחדת שנגדיר כאן, ולא בהורשה המובנית של Squeak. כלומר, מופע המחלקה OOPObject לא יופיע במערך הנ"ל). בנוסף, כל מחלקה כזו תכיל אוסף (כ-class-instance variable) של מופעים של המחלקה OOPParent שנגדיר בהמשך, כאשר כל מופע כזה יציין מחלקה ממנה יורשים + ה- Access Specifiers שלה.

כדי לממש את הקריאה ממופע של מחלקה מסוימת למתודות הנמצאות באחת ממחלקות הבסיס שלו, נממש מתודה שתתפוס את ההודעות הנ"ל, ונממש אלגוריתם חיפוש מתודות המתחשב ב-Access Modifiers.

הערה חשובה: בתרגיל זה נממש מנגנון הורשה מרובה שמתייחס רק למתודות, המנגנון שנממש יתעלם מירושה של שדות.



דוגמה 1: המחשת מנגנון ההורשה

בדוגמה לעיל ניתן לראות כי כל המחלקות (A,B,C) יורשות באופן סטנדרטי מ-OOPObject. כמו כן, מהגדרת מנגנון ההורשה המרובה ניתן להסיק כי A יורש מ-B ומ-C – <u>החצים הכתומים</u> מייצגים את **המופעים של OOPParent** שנמצאים במערך המשותף למחלקה, <u>והחצים הירוקים</u> מייצגים את **המופעים של תתי האובייקטים** של B ו-C בתוך המופע Instance A.

<u>כל המחלקות שתממשו בתרגיל זה יהיו תחת הקטגוריה OOP4!</u>

חלק א' – המחלקה OOPParent

כפי שתואר לעיל, המחלקה OOPParent תשמש אותנו במימוש מבנה הירושה החדש.

Instance Variables:

1. parentClass – משתנה מופע שיחזיק את המחלקה אותה העצם מייצג. מאותחל ל-Object.
2. inheritanceType – מחרוזת שתייצג את סוג ההורשה. הערכים האפשריים הם: 'protected' , 'private' , 'public'. מאותחל ל-'public'.

Instance Methods:

1. parentClass – מחזירה את ערך השדה parentClass.
2. inheritanceType – מחזירה את ערך השדה inheritanceType.
3. parentClass: aClass – מקבלת מחלקה ומציבה אותה לשדה parentClass.
4. inheritanceType: aString – מקבלת מחרוזת ומציבה אותה לשדה inheritanceType.

הערה: ניתן להניח שהקלט למתודות תקין.

חלק ב' – המחלקה OOPObject ומימוש המנגנון

נגדיר כעת את המחלקה OOPObject שתהווה בסיס לכל המנגנון.

משתנים:

1. superclasses – משתנה class-instance variable. משתנה זה יכיל Array של מופעים של OOPParent.
2. superclassInstances – משתנה instance variable שיכיל Array של מופעים של המחלקות המופיעות ב-superclasses.

מתודות:

1. ממשו את מתודת המחלקה הבאה:

subclass: aSubclassName **parentClasses:** anArray **instanceVariableNames:** instVarNames **classVariableNames:** classVarNames **poolDictionaries:** poolDictionaries **category:** aCategoryName

מתודה זו תאפשר הגדרת מחלקה שיורשת מ-OOPObject בירושה הרגילה של Squeak, תוך כדי הגדרת האבות הקודמים במנגנון החדש. מתודה זו זהה למתודה יצירת subclass של השפה פרט להוספת הפרמטר parentClasses: anArray. ב-anArray יהיו מופעים של OOPParent (שבפועל יגדירו את ההורשה המרובה). על המתודה לדאוג ליצירת המחלקה בעץ הירושה הרגיל של Squeak ולאתחל את שדה ה-superclasses שלה בהתאם לתוכן anArray. **ערך ההחזרה הינו המחלקה החדשה שנוצרה**. לשם אתחול ה-superclasses חפשו את המתודה instVarNamed:put:.

שימו לב: anArray **<u>יכול</u>** להיות מערך ריק. במידה והמתודה נקראת על מחלקה A ששונה מ-OOPObject יש לדאוג שההורשה תתאים למבנה החדש, כלומר, בהינתן מחלקה A שיורשת מ-OOPObject, אם המתודה נקראת בכדי להגדיר מחלקה B שיורשת מ-A עם parentClasses: anArray, יש להגדיר את B כך שתירש ישירות (במובן הרגיל של השפה) מ-OOPObject ומערך ה-superclasses של B יחזיק **בתא הראשון** את המחלקה A עם 'public' בתור ה-Specifier, ובשאר התאים את מופעי OOPParent שנמצאים ב-anArray. ניתן להניח שהקלט תקין ושהמחלקות במערך לא יחזרו על עצמן, ולא תהיה ירושה מעגלית. כמו כן, כל המחלקות ב-anArray יהיו כאלה שניתן ליצור מופע שלהם בקריאה ל-new, ללא זריקת חריגה.

2. **<u>דרסו</u>** את המתודה הבאה המוגדרת לראשונה ב-Behavior. **שימו לב:** אל תשנו את הקוד ב-Behavior!

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

על המתודה הדרוסה לדאוג לכך שכל המחלקות שיורשות מ-OOPObject **בירושה רגילה** יתאימו למבנה החדש. למשל אם המתודה נקראת בכדי להגדיר מחלקה A **היורשת מ-OOPObject**, יש לדאוג ש-A תאותחל עם מערך ריק ב-subclasses. לחילופין, אם המתודה נקראת בכדי להגדיר מחלקה B **היורשת מ-A** (ו-A יורשת מ-OOPObject) יש להגדיר את B כך שתירש ישירות מ-OOPObject, ו-A תהיה מחלקת אב במנגנון החדש עם 'public' Specifier'. לשם כך, השתמשו במתודה מהסעיף הקודם.

3. ממשו את מתודת **המחלקה** הבאה:

```
multInheritsFrom: aClass
```

המקבלת מחלקה ומחזירה true אם aClass הוא אב קדמון (במובן המנגנון החדש) של המחלקה הנוכחית ו-false אחרת. יש להתעלם מה-access specifiers במעלה עץ הירושה. ניתן להניח שמתודה זו **לא** תקרא על אובייקט מסוג OOPObject \ עם פרמטר מסוג OOPObject.

4. ממשו את מתודה **המחלקה** הבאה:

```
superclasses
```

המחזירה את ערך המשתנה superclasses.

5. ממשו את מתודת **המופע** הבאה:

```
initializeSupers
```

שתאתחל את superclassInstances למערך המכיל מופעים של מחלקות האב המוגדרות בשדה superclasses. **<u>יש לאתחל את האבות בסדר בו הוגדרו במערך.</u>**

6. הוסיפו את מתודת **המופע** הבאה:

postInitialize

המאפשרת (ע"י דריסה שלה) אתחול שמתבסס על האבות הקודמים המרובים, מכיוון שהיא נקראת אחרי שתת האובייקטים של האבות המרובים אותחלו.

מימושה (הדיפולטי):

```
postInitialize
        ^self
```

7. דרסו את מתודת **המופע** initialize ב-OOPObject למימוש הבא:

```
initialize
        ^self initializeSupers postInitialize
```

שימו לב: כל מחלקה דואגת קודם לאתחל את האבות שלה ורק לאחר מכן מאתחלת את עצמה, כך שהאתחול יכול להתבסס על המופעים של האבות הקודמים.

<u>הבהרה:</u> הכוונה באב קדמון הוא אחד ה-superclasses של המחלקה הנוכחית, או אב קדמון של אחד ה-superclasses שלהם.
<u>הערה:</u> מחלקות אב יכולות להיות **גם מחלקות שאינן יורשות מ-OOPObject! (בצורה עקיפה או ישירה)**

חלק ג' – חיפוש מתודות והפעלתן

א.
הגדירו את מתודת **המחלקה** הבאה:

classifyInheritedMethod: aSymbol

במחלקה OOPObject. המתודה מקבלת שם של מתודה (מופע של המחלקה Symbol או מחלקה היורשת ממנה) ומחזירה את אחת המחרוזות הבאות:

- **'public'** – אם המתודה נחשבת public במחלקה הנוכחית, ואין ambiguity כפי שיוגדר בהמשך.
- **'protected'** – אם המתודה נחשבת protected במחלקה הנוכחית, ואין ambiguity כפי שיוגדר בהמשך.
- **'private'** – אם המתודה נחשבת private במחלקה הנוכחית, ואין ambiguity כפי שיוגדר בהמשך.

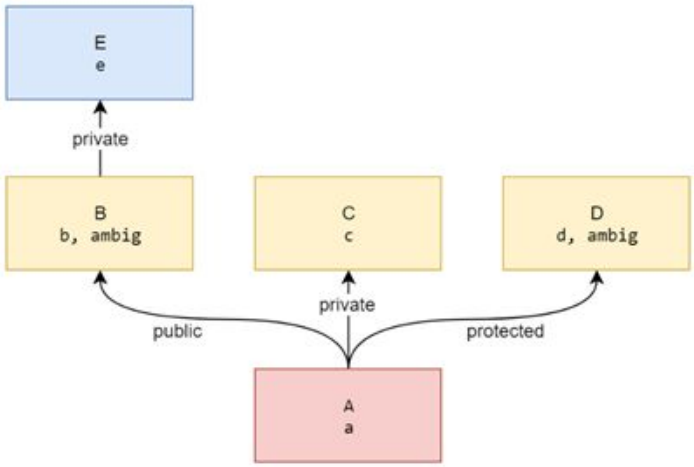
- **'ambiguous'** – אם ניתן להגיע למתודה בשם מסוים בכמה "מסלולים" שונים על גבי עץ הירושה. כזכור מחומר הקורס, מצב זה יכול לקרות באחד משני המקרים הבאים:
  - **התנגשות מקרית:** בהינתן מחלקה A אשר יורשת (לא בהכרח ישירות) מבסיסים B ו-C **<u>שונים</u>** אשר מגדירים שניהם מתודה ()f, כאשר עצם של A מנסה להפעיל את ()f נקרא להתנגשות התנגשות מקרית.
    כדי לתקן התנגשות מקרית, מספיק לשנות את שם אחת המתודות ב-B או ב-C.
  - **התנגשות מובנית**: בהינתן מחלקה A אשר יורשת (לא בהכרח ישירות) מבסיסים B ו-C אשר יורשים בעצמם ממחלקה D אשר מגדירה מתודה ()f, כאשר עצם של A מנסה להפעיל את ()f נקרא להתנגשות התנגשות מובנית.

התנגשות יכולה לקרות ללא קשר לנראות ההורשה, כלומר, בהינתן מתודה f שמוכרת ע"י יותר מאב קדמון אחד, אך באחד או יותר מהמסלולים היא נחשבת inaccessible (כפי שיוגדר בהמשך) הפעלת classifyInheritedMethod עליה תחזיר את הערך ambiguous!

- **'undefined'** – אם המתודה אינה מוגדרת באחד **מהאבות הקדומים** של המחלקה.

- **'inaccessible'** – אם המתודה קיימת באב קדמון כלשהו ואין ambiguity, אבל לא ניתן לקרוא לה בכלל מהמחלקה הנוכחית בגלל ירושת private במעלה המסלול (בעץ הירושה) לאב הקדמון (כלומר המחלקה הנוכחית יורשת ממחלקה אחרת שבשרשרת הירושה ממנה עד המחלקה שהגדירה את המתודה קיימת ירושת private ואין הגדרה חדשה של המתודה בשרשרת הירושה).
  <u>הערה:</u> עבור מתודות **שמוגדרות במחלקה עליה מופעלת המתודה** יש להחזיר 'undefined' (לפי ההגדרה).
  <u>הערה:</u> מתודה נחשבת public/protected/private/inaccessible לפי סוג ההורשה!

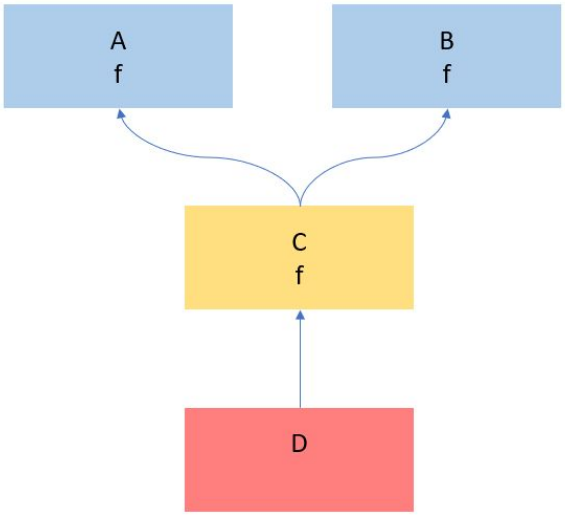
להמחשת פעולת המתודה, באיור הבא מתואר עץ ירושה מסוים, ומתחתיו טבלה ובה ערכי ההחזרה הצפויים עבור הפעלת המתודה מכל אחת מהמחלקות על כל אחת מהמתודות. **<u>מתחת לכל שם של מחלקה כתובות המתודות המוגדרות בה.</u>**



דוגמה 2: עץ הורשה עבור classifyInheritedMethod

	a	b	c	d	e	ambig
A	**undefined**	public	private	protected	inaccessible	**ambiguous**
B	undefined	**undefined**	undefined	undefined	private	**undefined**
C	undefined	undefined	**undefined**	undefined	undefined	undefined
D	undefined	undefined	undefined	**undefined**	undefined	**undefined**
E	undefined	undefined	undefined	undefined	**undefined**	undefined

דוגמא נוספת להבהרת ה-ambiguity:



בדוגמא הנ"ל A, B ו-C מגדירים את המתודה f. המתודה f תחשב ambiguous עבור D משום שיש לה 2 מסלולים שונים להגיע ל-f.

דגשים חשובים:

- כאשר מחלקה יורשת מ-OOPObject, המתודה classiftInheritedMethod **לא** צריכה להתחשב במתודות שנורשו כחלק מהירושה הרגילה של סקוויק, וצריך רק לבדוק מתודות שהוגדרו במחלקה (שלא נורשו אל המחלקה). כלומר לדוגמא, אם מחלקה A יורשת מ-OOPObject (שיורש בעצמו מ-Object), אז A לא צריכה "לדעת" על asString# המוגדר ב-Object. **שימו לב** שכאשר מחלקה אינה יורשת מ-OOPObject, אז המתודה classifyInheritedMethod **כן** צריכה להתחשב במתודות שנורשו כחלק מהירושה הרגילה של סקוויק, שכן כעת הן נחשבות כחלק מהמחלקה.
- אם מתודה שנחשבה private או inaccessible נדרסה על ידי מחלקה D נמוכה יותר בעץ ההורשה, אז המתודה הנדרסת צריכה להיחשב כאילו היא הוגדרה רק במחלקה D (עבור תת העץ של D).

ב.

הגדירו את מתודת **המופע** הבאה:

```
definingInstance: aSymbol
```

במחלקה OOPObject. המתודה תקבל שם של מתודה (מופע של המחלקה Symbol או מחלקה היורשת ממנה) ומחזירה את **המופע** של האב הקדמון שמגדיר את המתודה (או שמבין את המתודה על ידי הירושה הרגילה של סקוויק, גם אם לא הגדיר אותה בעצמו), או nil אם המתודה לא נמצאת או מוגדרת כ-'ambiguous' או 'inaccessible' עבור המחלקה. כלומר – המתודה מחפשת את המופע המתאים להפעלת המתודה בצורה רקורסיבית בין ה-superclassInstances.

הערות:

- המתודה מתעלמת מהמתודות שהוגדרו במחלקה הנוכחית, ומחזירה תשובה רק עבור מתודות שנורשו.
- לדוגמה, נתבונן במבנה הבא: מחלקה C יורשת מ-B שיורש מ-A (כל הירושות 'public'). ב-A מוגדרת המתודה foo וב-B מוגדרת bar. במבנה החדש למופע של C יש מערך superclassInstances בגודל 1 המכיל מופע של B, ולמופע הזה של B יש מערך superclassInstances בגודל 1 המכיל מופע של A. אם נריץ את השורות הבאות:

```
c := C new.
x := c definingInstance: #foo.
y := c definingInstance: #bar.
z := c definingInstance: #aaa.
```

אז x יכיל את המופע של A ו-y יכיל את המופע של B (תתי-אובייקטים של c), ו-z יכיל nil.

ג.

דרסו את מתודת **המופע** הבאה:

```
doesNotUnderstand: aMessage
```

במחלקה OOPObject. מתודה זו נקראת כאשר מנסים לשלוח הודעה לאובייקט, אך ההודעה אינה מוגדרת עבורו. כיוון שמבנה הירושה המוגדר בתרגיל אינו מוכר ע"י מנגנון חיפוש המתודות הסטנדרטי של Squeak, מתודה זו תקרא בכל פעם שנשלחת הודעה להפעלת מתודה שלא הוגדרה במחלקה **עצמה**. במימוש שלכם עליכם לחפש את המתודה בעץ הירושה המרובה. לאחר החיפוש יש להפעיל את המתודה אם אפשר, תוך התחשבות בהרשאות הגישה, כלומר האם למחלקה ששלחה את ההודעה יש הרשאות לגשת למתודה בהתאם ל-Access Specifiers בירושה.

ניתן להניח שלא יוספו מתודות למחלקות במנגנון החדש עם שמות המוכרים ל-OOPObject.

סדר הפעולות שעליכם לבצע ב-doesNotUnderstand הוא:

1. מצאו את המחלקה ששלחה את ההודעה. בכדי לעשות זאת יש לבצע את השורה הבאה:

   ```
   thisContext client class
   ```

2. אם נמצאה מתודה מתאימה (ה-Specifier הוא public, protected או private) קבעו אם המחלקה **ששלחה** את ההודעה רשאית להפעיל את המתודה אצל המחלקה **המקבלת**.
   a. אם כן, יש לשלוח את ההודעה למופע של תת האובייקט בו מוגדרת המתודה. השתמשו ב-definingInstance.
   b. אחרת (המתודה היא undefined, ambiguous או inaccessible, או שלמחלקה ששולחת את ההודעה אין הרשאות להפעיל את המתודה), יש לזרוק חריגה באמצעות מתודת **המחלקה** הבאה (שעליכם להוסיף ל-OOPObject):

```
throwSender: senderName fails: methodName inClass: receiverName because: reason
| str |
	str := senderName, ' cannot send ',methodName asString,' to ',receiverName,' because: '
,reason.
	AssertionFailure signal: str.
```

תיאור פרמטרים:

- **senderName** – מחרוזת. שם המחלקה שניסתה לשלוח את ההודעה.
- **methodName** – מופע של Symbol (או מחלקה שיורשת ממנה). הסלקטור של המתודה.
- **receiverName** – מחרוזת. השם של המחלקה שההודעה נשלחת אליה.
- **reason** – מחרוזת. סיווג המתודה במחלקה המקבלת.

<u>הערות:</u>

- מי רשאי להפעיל מתודה: נניח שמחלקה B שלחה הודעה למחלקה A. אם המתודה נחשבת **private** ב-A אז B יכולה להפעיל אותה אם ורק אם **A==B**. אם המתודה נחשבת **protected** ב-A אז B יכולה להפעיל אותה רק אם **B יורשת מ-A** או **A==B**. אם המתודה נחשבת **public** אז **כל מחלקה B** יכולה להפעיל את המתודה.
- כדי לקבל את הסלקטור של המתודה ש-doesNotUnderstand מייצגת, שלחו את ההודעה selector ל-aMessage.
- אפשר להניח שערכי חזרה של המתודות שאותן אתם מפעילים לא יבדקו
- למתודות **יכולות** להיות פרמטרים. מומלץ לבדוק את המחלקה של aMessage בשביל לראות איך עושים זאת.
- **<u>חשוב</u>**: לא חייבים להשתמש ב-thisContext לשום חלק אחר בתרגיל! אם בכל זאת המימוש שלכם משתמש בו, בצעו fileout לקוד שלכם לפני ניסיון הרצה. גישה לא נכונה ל-thisContext תביא לקריסת ה-VM ולאובדן כל הקוד שלא ייצאתם.

Crash Course: יצירת מתודות ומחלקות בזמן ריצה - על מנת שתוכלו לבדוק את המימוש שלכם

כידוע, מודל 5 הרמות של השפה נותן לנו כמתכנתים שליטה מלאה על המחלקות המוגדרות – כולל האפשרות ליצור אותן בזמן ריצה. כדי לבדוק את המימוש שלכם מומלץ מאוד לייצר מחלקות בזמן ריצה.

כדי ליצור מחלקה B היורשת ממחלקה A שלחו את ההודעה `subclass` ל-A (או את `subclass` שמימשנו כאן בתרגיל):

```
A subclass: #B instanceVariableNames: '' classVariableNames: '' poolDictionaries:
'' category: ''
```

כדי "לקמפל" מתודה חדשה למחלקה A שלחו את ההודעה `compile` ל-A:

```
A compile: aString
```

מתודה מקבלת מחרוזת המכילה את המימוש המלא של המתודה (כלומר מכילה את החתימה שלה, את הגדרות הארגומנטים שלה ואת המימוש שלה) לפי התבנית הבאה:

```
messageSelectorAndArgumentNames
       | temporary variable names |
       statements
```

טיפים שימושיים

- יתכן שיהיה לכם יותר קל לפתור את התרגיל אם תתייחסו לחיפוש המתודות במעלה הירושה בתור חיפוש בגרף.
- לפני שאתם ניגשים לפתרון, מומלץ לעבור שוב על תרגול מודל האובייקט ב-Squeak 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-Browser. 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-Browser. מומלץ להתחיל ב-Behavior ולחפש בו מתודות לפי הקטגוריות.
- 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. מותר להגדיר שדות כרצונכם.
- **אין להדפיס דבר לפלט (Transcript). אם אתם מדפיסים לצורך בדיקות, הקפידו להסיר את ההדפסות לפני ההגשה**.
- אין צורך לבצע בדיקות טיפוסים על הארגומנטים של המתודות.
- 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) או ב-Pharo (שפה אשר דומה מאוד ל-Smalltalk).

הוראות הגשה

- **בקשות לדחייה, מכל סיבה שהיא, יש לשלוח למתרגל האחראי על הקורס (אופיר) במייל בלבד תחת הכותרת HW4 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 OOP4_<ID1>_<ID2>.zip המכיל:
- קובץ בשם readme.txt המכיל שם, מספר זיהוי וכתובת דואר אלקטרוני עבור כל אחד מהמגישים בפורמט הבא:

Name1 id1 email1
Name2 id2 email2

- קובץ הקוד: **OOP4.st**. על הקובץ להכיל רק את מימוש המחלקות שהוזכרו בתרגיל 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 וכדומה)

בהצלחה!