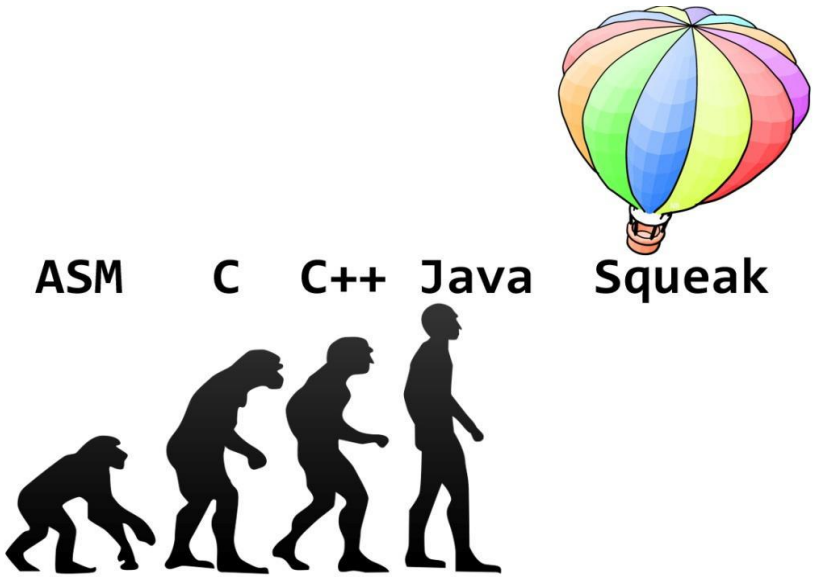
236703 – תכנות מונחה עצמים, אביב 2022-2023

תרגיל בית 3: Squeak מתקדם



כללי

1. מועד ההגשה: **25/05/2023**, ההגשה היא בזוגות.
2. קראו היטב את ההוראות במסמך זה.
3. אחראי על התרגיל: **עידו**. שאלות על התרגיל יש לשלוח למייל idomagner@campus עם הנושא: "236703 HW3".

   שאלות אדמיניסטרטיביות (כגון מילואים) יש להפנות לג'וליאן, במייל, תחת אותה כותרת.
4. **לפני הרצת קוד, שמרו את השינויים שעשיתם בקובץ נפרד על-ידי fileout (קליק ימין).**

   אנו ממליצים לכם לבצע שמירות אלו באופן תכוף כיוון שלSqueak יש נטיה לקרוס בזמנים שאינם צפויים.
5. **במידה וחלקי קוד נמחקים לכם מה workspace-(כולל מחלקות או מתודות שנמחקות), אתם יכולים לנסות לשחזר את הקוד מקובץ log שנמצא בתיקייה: Contents\Resources**
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. כדי 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, הכוללות יצור מחלקות ויירוט ההודעות הנשלחות לאובייקט. בעזרת כלים אלו נממש Enums.

שימו לב –

- מכיוון שאנחנו לא רוצים לשנות את המנגנונים המובנים של **Squeak**, נממש מחלקה משלנו **OOPEnum** שתתמוך בשינויים של תרגיל בית זה.
- כמה הסברים טכניים, טיפים ורמזים מפורטים בהמשך, קראו אותם לפני שאתם ניגשים למימוש.
- כאשר אתם נדרשים לדרוס מתודה, עליכם להגדיר אותה במחלקה שלכם, ולא לשנות את הקוד המקורי של המתודה היכן שהיא מוגדרת במקור!
- כדי לשלוח שגיאות לכל אורך התרגיל השתמשו בשורה הבאה:

```
AssertionFailure signal: aMessage.
```

כאשר aMessage היא מחרוזת המכילה את הודעת השגיאה.

- כל המחלקות שתממשו בתרגיל זה יהיו תחת הקטגוריה **OOP3**

מה זה Enum (תזכורת)?

ספציפית **Enum** בשפת **Java**, כמו שראינו בתרגול.

המאפיינים של **Enum** (שנרצה לאפשר):

1. מס' ערכים מוגדר וקבוע מראש
2. final – כלומר לא יוכלו לרשת מהEnum
3. אפשר להגדיר שדות
4. המופעים נוצרים פעם אחת, אופציונלית באמצעות בנאים שמוגדרים עבורם במיוחד
5. אפשר להגדיר מתודות
6. הערכים עצמם:
    a. הגישה אליהם תהיה דרך הEnum עצמו
    b. יכולים לדרוס מתודות של הEnum

איך ניצור Enums?

אז איך המנגנון שלנו הולך לעבוד?

מחלקת האב הגנרית של כל ה-Enums שלנו תהיה OOPEnum, ונבנה אותה כך שכל מחלקה היורשת ישירות מ-OOPEnum היא Enum.

OOPEnum שאנחנו מגדירים בתרגיל תגדיר לנו את הפרוטוקול של כיצד Enum בשפה אמור להתנהל – היא לא מייצגת Enum בעצמה! לכן חלק מהמתודות שנגדיר לה הן למעשה לשימוש המחלקה שתירש ממנה (הEnum עצמו).

חשוב להבין כי אנחנו מתעסקים בתרגיל זה עם ירושה שהיא לא סטנדרטית, ולא מתאימה לחלוטין לאבסטרקציות שאנחנו לומדים בקורס – Enums הם מקרים ספציפיים, המופעים של Enum הם גם הערכים שלו והם בעצם יורשים מה-Enum עצמו בדרך מוזרה – לכן גם אנחנו "נרמה" קצת את המערכת.

כל מחלקת Enum במנגנון שלנו תשמור את המופעים שלה (כלומר הערכים שלה) בתוך שדות של המחלקה, שיקבעו בעת יצירת המחלקה.

האתחול של המופעים יתחלק לשתי אפשרויות – אתחול עצל (lazy-initialization), ואתחול מיידי (eager-initialization), לפי דגל המועבר בעת יצירת המחלקה.

יהיו לEnum מתודות ייעודיות לגישה לערכים, אשר בעת אתחול עצל בנוסף להחזרת הערכים, יאתחלו אותם באופן עצל– הן יחזירו את המופע אם קיים, ואם לא יאתחלו אותו וישמרו בשדה המתאים (בדומה לסינגלטון בתרגול).
במידה והאתחול הוא eager, המופעים יווצרו כבר עם הגדרת המחלקה.

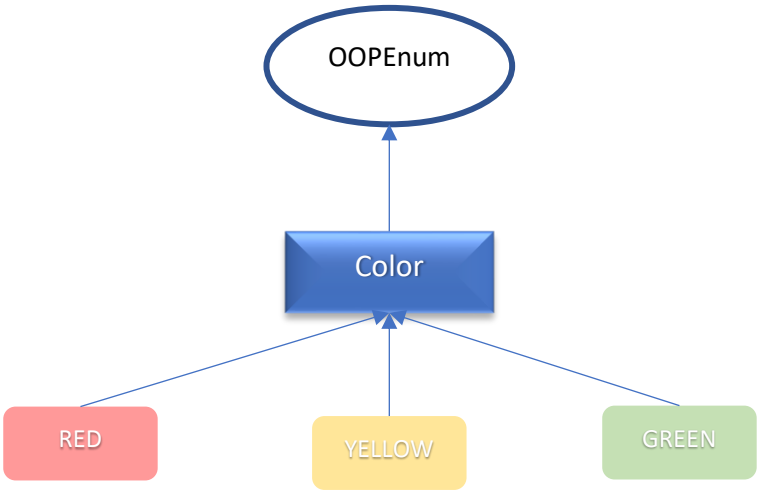
בצורה זו נוכל לוודא כי הגישה לערכים היא רק דרך מחלקת הEnum עצמה ואכן יש מספר ערכים סופי וקבוע מראש – נשנה את המתודה new, ונאתחל את הערכים בעקיפין, בדומה לנעשה בMetaclass.

משום שהערכים של הEnum הם למעשה המופעים שלו, ונרצה להגדיר להם שדות ומתודות, אך גם לאפשר להם לדרוס את המתודות הללו, נבצע זאת בעזרת ירושה.
הEnum יהיה מחלקת האב, וניצור לו (אוטומטית) מחלקות "אנונימיות" שיורשות ממנו ומגדירות את ההתנהגות של כל אחד מהערכים, והערכים עצמם יהיו מופעים של המחלקות הללו.

בנוסף נחסום את הירושה מOOPEnum, הEnum שהמשתמש ייצור והמחלקות האנונימיות שלו (תראו בהמשך כיצד), כך שהם יהיו final.

<u>נמחיש את מנגנון הEnums בדוגמה:</u>

עבור Enum בשם Color שיש לו את הערכים YELLOW, RED ו-GREEN (כמו בתרגול):



איך יוכלו להשתמש במנגנון שלנו?

כזכור לכם, בSqueak ה"קונסטרקטורים" שאנחנו מכירים שמעדכנים את השדות של האובייקט, הם למעשה מתודות רגילות שיש לקרוא להן בנפרד מאתחול האובייקט – האתחול הדיפולטי שלו מתבצע באמצעות initialize.

כדי לאתחל את המופעים עם קונסטרקטורים שונים, יהיה על המשתמש בEnum (לא במסגרת התרגיל! זכרו – אנחנו רק מגדירים פה פרוטוקול. אך כך תצטרכו לנהוג בטסטים) ליצור את הקונסטרקטורים בעצמו – להגדיר אותם בתור מתודות מופע בEnum שיורשו למחלקות של הערכים, ומבצעות השמה לשדות המתאימים.

כלומר כל שינוי של השדות הפנימיים של הערכים יהיה באחריות המשתמש בEnum, אך משום שהערכים הם סינגלטונים, לשנות אותם פעם אחת תספיק.

כדי להגדיר שדות ומתודות לEnum, יהיה על המשתמש בEnum להגדיר אותם במחלקת הEnum, והם יורשו למחלקות של הערכים אוטומטית. כדי לדרוס מתודה עבור מופע ספציפי, המשתמש יוכל לדרוס אותה במחלקה האנונימית של הערך.

Crash Course: יצירת מתודות ומחלקות בזמן ריצה

כידוע, מודל 5 הרמות של השפה נותן לנו כמתכנתים שליטה מלאה על המחלקות המוגדרות – כולל האפשרות ליצור אותן בזמן ריצה. בתרגיל זה תידרשו כמה וכמה פעמים ליצור מחלקות חדשות ולהוסיף מתודות למחלקות בזמן ריצה.

כדי ליצור מחלקה B היורשת ממחלקה A בדרך הרגילה שולחים את ההודעה subclass ל-A:

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

המתודה ... :subclass מוגדרת במחלקה Class.

שימו ♥: שם המחלקה הוא מסוג Symbol (מוגדר ע"י סולמית #) – לא String רגיל! כאשר אתם יוצרים מחלקות, דאגו להגדיר את השם שלהם בתור Symbol, זה גם הטיפוס שיועבר למתודה ... :subclass בארגומנט הראשון.

בתרגיל זה אנחנו מנסים לעקוף את הדרך הרגילה ליצור מחלקות למטרות התרגיל (לוודא כי לא ניתן יהיה לרשת מהמחלקה שלנו), שימו לב לכך.

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

```
A compile: aString
```

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

```
messageSelectorAndArgumentNames

        | temporary variable names |

        statements
```

חלק א

הגדירו את המחלקה הבאה, שתשמש כמחלקת האב שלנו (כמו המחלקה Enum ב-Java):

OOPEnum

Class-instance variables:

1. **`values`**

   שדה זה יכיל מערך של השמות של כל הערכים של הEnum.

2. **`lazyInitialization`**

   דגל המציין האם אתחול המופעים נעשה בצורה עצלה או לא.

שימו לב! Class-instance variables הם שונים מ-class variables (ודאו כי אתם מבינים מדוע).
ניתן להגדיר class-instance variable ע"י לחיצה על כפתור הclass ושינוי השדות של instanceVariableNames של OOPEnum class.

Class methods:

הגדירו את מתודות ה**מחלקה** הבאות:

1. **`new`**

   דרסו מתודה זו, כך שתשלח את הודעה השגיאה:
   "You can't create an instance of <thisClassName>"
   כאשר <thisClassName> הוא שם המחלקה הנוכחית.
   בצורה זו לא יהיה אפשרי ליצור מופעים של המחלקה וגם של כל המחלקות שירשו ממנה. (נראה בהמשך כיצד אנחנו, בתור המתכננים של המחלקה, יוצרים ידנית את ה"מופעים" (הערכים של הEnum))

2. **`subclass:`** `aSubclassName` **`instanceVariableNames:`** `instVarNames` **`classVariableNames:`** `classVarNames` **`poolDictionaries:`** `poolDictionaries` **`category:`** `aCategory`

   דרסו מתודה זו, כך שתשלח את הודעה השגיאה:
   "You must specify <aSubclassName>'s values"
   כאשר <aSubclassName>הוא הארגומנט של הפונקציה.
   בצורה זו נוודא כי המחלקות שירשו מהמחלקה שלנו (כלומר ה-Enum-ים בשפה) יוכלו להיווצר רק בדרך שאנו מגדירים.

3. **subclass:** aSubclassName **values:** valuesArr **lazyInitialization:** aBoolean **initialize:** initializeAsString **instanceVariableNames:** instVarNames **classVariableNames:** classVarNames **poolDictionaries:** poolDictionaries **category:** aCategoryName

ממשו מתודה זו בהתאם לתיאור בהמשך. זו למעשה המתודה המרכזית של התרגיל!

הסבר: מתודה זו תאפשר הגדרת מחלקות שיורשות מ-OOPEnum – כלומר להגדיר את טיפוסי ה-Enums עצמם. היא מחליפה את המתודה היוצרת subclass של השפה, וזהה לה פרט להוספת הפרמטרים lazyInitialization: aBoolean , values: valuesArr ו- initialize: initializeAsString.

valuesArr – יישמר בclass-instance variable המתאים, ויכיל את שמות הערכים של ה-Enum (בתור מחרוזות).

aBoolean – דגל המסמן האם אנחנו רוצים לאתחל את המופעים(הערכים) באופן עצל. אם הדגל כבוי (false), ניצור את המופעים של הEnum (הערכים שלו) כעת (במתודה זו) ונשמור אותם בclass-instance variable המתאים להם. אחרת אם הדגל דולק (true) ניצור כל מופע בגישה הראשונה אליו.

initializeAsString – מחרוזת המתארת את הגדרת (ומימושה המלא!) המתודה initialize (אשר מאתחלת את המופעים של המחלקה ומהווה קונסטרקטור דיפולטיבי) שיש להוסיף כמתודת **מופע** של מחלקת הEnum החדשה.

למשל מימושה הדיפולטי:

```
initialize
	^self
```

משום שאנחנו מתעסקים עם ירושה מוזרה, נכופף הפעם מעט את הכללים – נרצה שמתודה זו (subclass) תעבוד רק עבור OOPEnum, כדי לוודא שלא יוכלו לרשת מה-Enums (שאנחנו יוצרים באמצעות מתודה זו).

תיאור כללי למתודה:

אם המתודה נשלחת למחלקה שהיא לא OOPEnum, שלחו את הודעה השגיאה:
"<thisClassName> is final! It can't be inherited"
כאשר <thisClassName> הוא שם המחלקה הנוכחית.

המתודה יוצרת מחלקה חדשה היורשת מOOPEnum (זה בעצם הEnum), המכילה את הפרמטרים שהועברו למתודה.
המחלקה החדשה תכיל class-instance variable עבור כל ערך של הEnum, בעל אותו שם כמו

בvaluesArr. בנוסף initialize תהיה מוגדרת במחלקה החדשה כמו בפרמטר initializeAsString.

צרו למחלקה החדשה מחלקות "אנונימיות" שיורשות ממנה ומגדירות את ההתנהגות עבור כל ערך של הEnum. מחלקות אלו יהיו כאילו-אנונימיות, במובן שנגדיר להן שם מיוחד: כל מחלקה כזו תיקרא לפי התבנית __EnumName_ValueName (שימו לב שיש בסוף פעמיים underscore (התו '_')), כאשר EnumName הוא השם של הEnum (aSubclassName) ו-ValueName הוא השם של הערך הספציפי של הEnum. יש ליצור את המחלקות במתודה זו בלי קשר לדגל lazyInitialization.

<u>לדוגמה:</u>

עבור הColor Enum, השם של המחלקה האנונימית המייצגת את הערך GREEN יהיה: __Color_GREEN.

למחלקה החדשה יהיו מתודות גישה עבור כל ערך של הEnum – כל מתודה תיקרא בשם הערך המתאים לה ובלי ארגומנטים.
המתודות האלו יהיו מתודות **מחלקה**.

<u>השימוש במתודות יהיה כך:</u> למשל כדי לקבל את המופע RED של Color נשלח את ההודעה **RED** למחלקה Color:

```
redInstance := Color RED.
```

יצירת המופע של הערך תיעשה באמצעות השורה:

```
ValueClassName basicNew initialize
```

כאשר ValueClassName הוא השם של המחלקה "האנונימית" של הערך (למשל __Color_RED).

המתודה תחזיר את המחלקה החדשה.

4. **compile:** aSourceCode

תפקיד מתודה זו הוא להדר את הקוד aSourceCode (להוסיף מתודת מופע שהחתימה והמימוש שלה מופיעות ב-aSourceCode למחלקה שלה שלחו את ההודעה).

<u>דרסו</u> מתודה זו, כך שתהדר את הקוד aSourceCode , אך תוודא עבור כל מתודה שמנסים להוסיף ל**מחלקה אנונימית** (כזו שמייצגת ערך של הEnum), שהיא מתודה הקיימת במחלקת הEnum עצמה (כלומר שכל מתודה במחלקה האנונימית דורסת מתודה במחלקת הEnum), אחרת תיזרק השגיאה:

"<MethodSelector> does not override a method from <EnumName>"

כאשר <MethodSelector> היא חתימת המתודה שמנסים להדר **בלי שמות הארגומנטים**, ו-<EnumName> הוא שם מחלקת הEnum.

לדוגמה: אם עבור Color מוגדרת המתודה **foo:** anInt, הקוד הבא ידרוס את המתודה במחלקה של RED:

```
Color_RED__ compile: 'foo: x
        ^x.'.
```

אך עבור ניסיון להדר את המתודה **bar:** aString **zap:** aChar תיזרק השגיאה:

"bar:zap: does not override a method from Color"

משום שהמתודה לא מוגדרת אצל Color:

```
Color_RED__ compile: 'bar: aString zap: aChar
        ^(aString, (aChar asString)).'.
"error"
```

הערה: מתודה דורסת מתודה על סמך חתימתה ובלי התחשבות בשמות הארגומנטים, לכן **foo:** anInt ו- **foo:** x מייצגות את אותה החתימה – **foo:**.

חלק ב

Switch

כעת נוסיף את הפונקציונליות של switch-case לEnum שלנו, בדומה לאופן פעולתו על Enum בJava כפי שראינו בתרגול.

נרצה שהswitch שלנו יתנהג כמו שבswitch-case על Enum בJava - ה-cases יכולים להיות מוגדרים גם רק על תת קבוצה של ערכי ה-Enum. למשל עבור Color נוכל להגדיר בJava את הswitch-case הבא:

```
//Java
Color c;
...
switch(c){
        case RED:
                System.out.println("RED");
                break;
        case YELLOW:
                System.out.println("YELLOW");
                break;
        default:
                System.out.println("GREEN");
        }
```

כך שה-cases מוגדרים רק עבור RED ו-YELLOW (ואילו GREEN משתמש בcase הדיפולטיבי).

הוסיפו למחלקה **OOPEnum** את מתודת ה**מחלקה** הבאה:

`addSwitch`

ממשו מתודה זו, כך שהיא תוסיף את הפונקציונליות של switch לEnum שלנו.

על מתודה זו להיות מופעלת רק על מחלקה שהיא Enum (כלומר יורשת מ-OOPEnum, לא כולל OOPEnum עצמה! אפשר להניח כי היא לא תופעל על המחלקות האנונימיות שיורשות מה-Enum),
אחרת תישלח השגיאה:

"can't add switch functionality to non-Enum class <classname>"

כאשר <classname> הוא שם המחלקה הנוכחית.

המתודה תוסיף את מתודות ה**מופע** הבאות למחלקה הנוכחית (ה-Enum, לא ל-OOPEnum!):

שמות המתודות יהיו לפי התבנית:

- אם המתודה מייצגת switch-case עבור תת קבוצה של ערכים (**שהיא לא ריקה!**):

```
Switch_case_<val1>: <val1>Block case_<val2>: <val2>Block ... default: defaultBlock
```

כאשר <val> הם הערכים בתת הקבוצה, והם מסודרים לפי הסדר בו הועברו למתודה subclass (בארגומנט valuesArr).

- אם המתודה מייצגת switch-case עבור כל הערכים, שם המתודה יהיה כמו בנקודה הקודמת, בלי הdefault בסוף.
- לא תהיה מתודה עבור תת קבוצה ריקה של ערכים. חייב להיות case לפחות לערך אחד.

המתודות יפעלו כך:

אם קיים case המתאים לערך של הEnum, נפעיל את הבלוק המוגדר עבורו. אחרת נפעיל את בלוק הdefault.

דוגמאות להתנהגות הרצויה של המתודות:

עבור Color שערכיו הם RED, YELLLOW ו-GREEN:

דוגמאות לשמות של המתודות:

- **switch_case_RED**: REDBlock **case_YELLOW**: YELLOWBlock **case_GREEN**: GREENBlock
- **switch_case_RED**: REDBlock **case_YELLOW**: YELLOWBlock **default**: defaultBlock
- **switch_case_RED**: REDBlock **default**: defaultBlock

וכך עבור כל תת קבוצה של ערכים (חוץ מריקה).

והן יתנהגו כך:

```
ourEnum := Color RED.

ourEnum switch_case_RED: [Transcript show: 'RED'; cr.] case_YELLOW: [Transcript show: 'YELLOW'; cr.] case_GREEN: [Transcript show: 'GREEN'; cr].

ourEnum switch_case_YELLOW: [Transcript show: 'YELLOW'; cr.] case_GREEN: [Transcript show: 'GREEN'; cr] default: [Transcript show: 'Default color: RED'; cr].
```

במקרה זה יודפס:

RED
Default color: RED

רמז: חשבו על אופן המימוש של המתודה **ifTrue: ifFalse:** במחלקות Boolean, True ו-False.

EnumDictionary

כעת נוסיף לשפה Collection מיוחד עבור Enums, בדומה ל-EnumMap בJava שראינו בתרגול.

EnumDictionary יתנהג כמו Dictionary, ותהיה לו כל הפונקציונליות הקיימת של Dictionary, עם טוויסט – אילוצים על הטיפוסים של המפתחות שלו. כלומר, עבור כל מופע של EnumDictionary מוגדר ה-Enum שהוא עובד איתו, וכל המפתחות שלו חייבים להיות מופעים (ערכים) של ה-Enum הזה.

למשל EnumDictionary של Color יכול להכיל בתור מפתחות רק את YELLOW, RED ו-GREEN.

הגדירו את המחלקה EnumDictionary שתירש מ-Dictionary (שימו לב – הגדירו אותה בקטגוריה OOP3):

:instance variables

1. **enumClass**
   ישמור את מחלקת הEnum של המפתחות שלו.

:instance methods

1. **`initialize`**
   דרסו מתודה זו - זו מתודה לאתחול האובייקט.
   על השדה enumClass לא להכיל אף טיפוס.

2. **`setEnum:`** `anEnumClass`

   ממשו מתודה זו, כך שתקבע את טיפוסי המפתחות של המילון. אם טיפוסי המפתחות כבר נקבעו, היא תשלח את השגיאה:

   “the Dictionary’s Enum is already defined with <enumClassName>”

   כאשר <enumClassName> הוא שם ה-Enum שהוגדר למילון.

   אם הטיפוס שהתקבל אינו טיפוס Enum, היא תשלח את השגיאה:

   “<className> is not an Enum”

   כאשר <className> הוא שם הטיפוס שהתקבל.

3. **`getEnum`**

   ממשו מתודה זו, כך שתחזיר את ה-Enum המוגדר להיות טיפוס המפתחות במילון.

4. **`add:`** `anAssociation`

   דרסו מתודה זו, כך שתוכל להוסיף למילון רק זוגות שהמפתח שלהם הוא אחד המופעים (הערכים) של ה-Enum (השמור בשדה **enumClass**).

   במידה והמפתח הוא לא אחד המופעים של ה-Enum, שלחו את השגיאה:

   “this EnumDictionary only accepts <enumClassName>’s values as keys”

   כאשר <enumClassName> הוא שם ה-Enum השמור ב-**enumClass**.

:class methods

1. `anEnumClass` **onEnum:**

ממשו מתודה זו, כך שתאתחל מופע חדש של EnumDictionary, ותקבע את טיפוסי המפתחות שלו, ותחזיר את המופע.

אם הטיפוס שהתקבל אינו טיפוס Enum, היא תשלח את השגיאה:

"`<className> is not an Enum`"

כאשר <className> הוא שם הטיפוס שהתקבל.

טיפים שימושיים והערות

- לפני שאתם ניגשים לפיתרון, מומלץ לעבור שוב על תרגול מודל האובייקט ב-Squeak ובפרט על התרשים המציג את מודל 5 הרמות בשפה.
- כמו כן מומלץ לעבור שוב על התרגול של ה-Enums בשפת Java על מנת לתפוס טוב את הקונספט של המנגנון שאנחנו מנסים לממש בתרגיל.
- 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.
  הצפוי הוא שתבינו בעצמכם איך לבצע את הפעולות הנדרשות בSqueak.
- מומלץ בשביל ההבנה שלכם ועל מנת לדאבג את הקוד (והכיף שלכם!) לנסות ליצור Enums בעצמכם באמצעות המנגנון, ולראות את המחלקות שנוצרות באופן אוטומטי (הנאה מובטחת!).
  שימו לב לשמות של הEnums שאתם יוצרים, שלא "ידרסו" מחלקות קיימות בSqueak (למשל Color היא כבר מחלקה קיימת..)
- 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. אין להוסיף שדות.
- **<u>אין להדפיס דבר לפלט (Transcript). אם אתם מדפיסים לצורך בדיקות, הקפידו להסיר את ההדפסות לפני ההגשה.</u>**
- אין צורך לבצע בדיקות טיפוסים על הארגומנטים של המתודות, למעט אלו הנדרשות מפורשות.
- באתר הקורס תחת התרגיל מופיע קובץ OOP3SamleTests.st. זהו קובץ "טסטים" שמטרתו העיקרית היא להראות לכם דוגמאות שימוש למחלקות שתממשו בתרגיל זה כדי שתוכלו להבין את השימוש התקין במחלקות, ולא כדי לבדוק אם המימוש שלכם עובד – אל תסתמכו עליו לשם כך! הקובץ לא בודק תקינות, רק מדפיס כמה דברים אל הפלט, והרצתו לא אמורה לזרוק שגיאות.
  כדי לראות את ההדפסות שלו פתחו את חלון הTranscript לפני הרצתו.
- 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 (בעז) 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 באתר הקורס (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 בפורמט הבא:

    Name1 id1 email1

    Name2 id2 email2
  - קובץ הקוד: OOP3.st. על הקובץ להכיל רק את מימוש המחלקות שהוזכרו בתרגיל ומתודות לצורך פתרון התרגיל (תחת הקטגוריה OOP3). **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)
- Memes יתקבלו בברכה!

בהצלחה!