נקודות "התנגשות" בין
תכנות פרוצדוראלי לתכנות מונחה עצמים
בשימוש בשפות התכנות החדשות

ד"ר נוע רגוניס

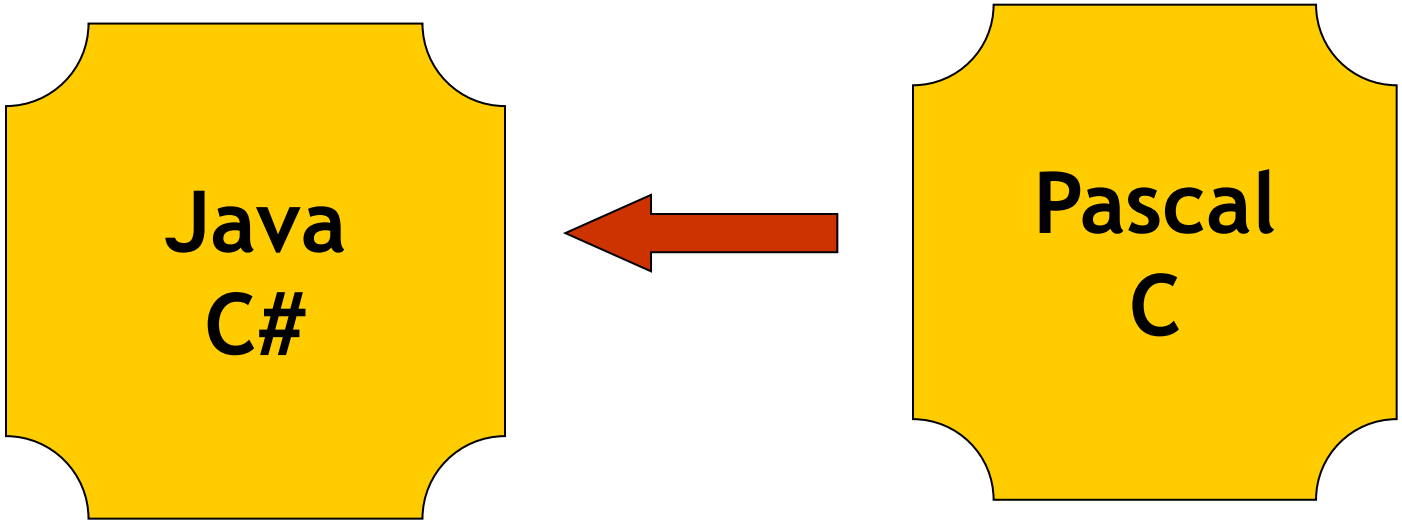
מכון ויצמן למדע רחובות

מכללת בית ברל

כנס המורים הארצי השישי למורי מדעי המחשב וטכנולוגיות מידע
אחוה – המכללה האקדמית לחינוך
חנוכה תשס"ו

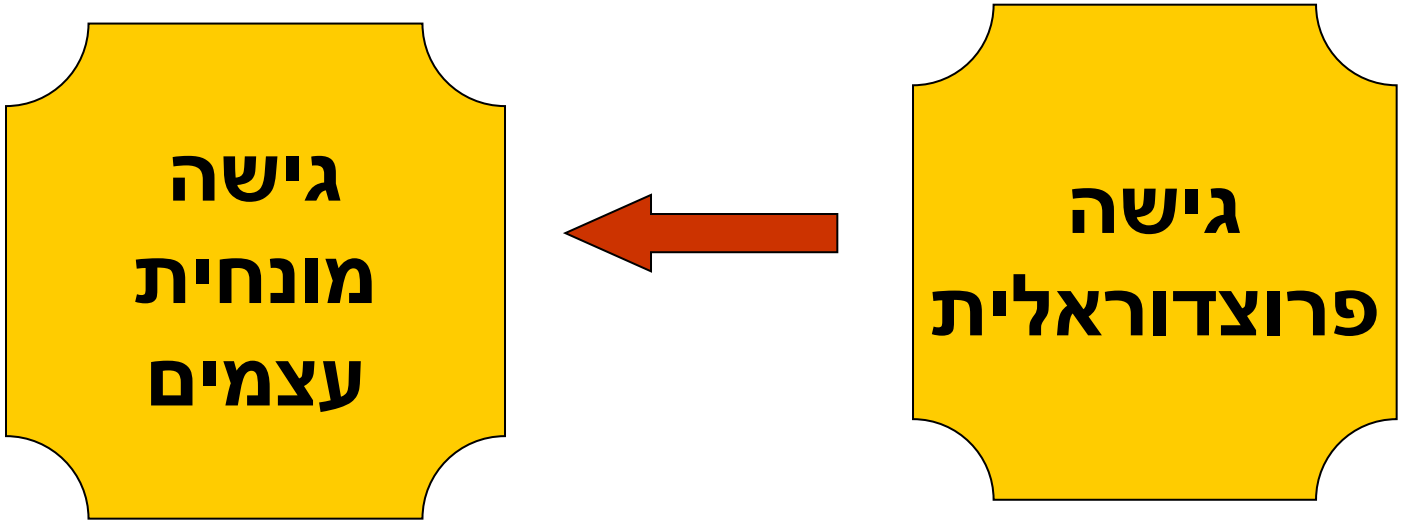
הקדמה

- אנו נמצאים בעיצומו של תהליך החלפה של שפות התכנות בהוראת תכניות הלימודים "יסודות מדעי המחשב" ו"עיצוב תוכנה".



הקדמה

- פרט מן ההבדלים התחביריים יש הבדלים עקרוניים **במבנה השפות**.



הקדמה

- בהוראת "יסודות מדעי המחשב" הכוונה להשאיר את הציר המרכזי והלא משתנה של תחום הדעת – **אלגוריתמיקה**, תוך שימת דגש על ניתוח בעיה, כתיבת אלגוריתם לפתרונה ומימושה בשפת התכנות.

- בהוראת "עיצוב תוכנה" יוקנה הבסיס של תכנות מונחה עצמים ומימוש טיפוסי הנתונים המופשטים המרכזיים (רשימה, מחסנית, עץ) **יהיה בגישה מונחית עצמים**.

מטרות ההרצאה

- בהרצאה, אדגים את ההשתקפות השונה של מספר נושאים מתוך תוכניות הלימודים, בהקשר של מימושם בשפות שנועדו לתכנות פרוצדוראלי לעומת מימושם בשפות שנועדו לתכנות מונחה עצמים.

- המטרה היא להציף את המקומות השונים בהם יש **"התנגשויות"** כדי להביא למודעות שלנו המורים לנקודות רגישות.

"התנגשויות" על שום מה?

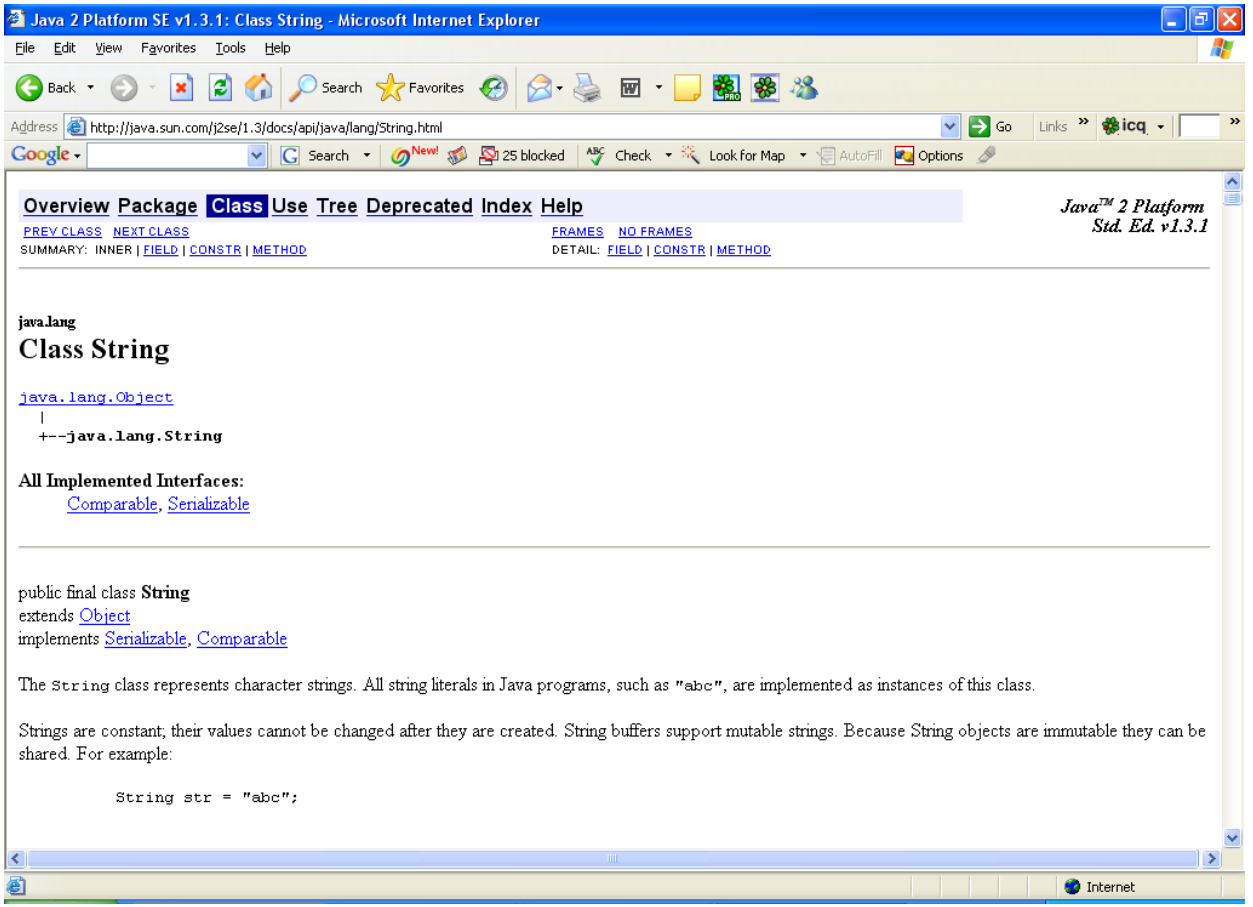
- השפות החדשות מטבען עשויות ליצור בלבול אצל הלומד, בהקשר לתפיסת תכנות מונחה עצמים (String).
- השפות החדשות מציעות כלים שונים בתכלית למימוש סעיפים של תכנית הלימודים (רשומות).
- מימוש התואם רק את הנושאים הכלולים בתכניות הלימודים, משבש לפעמים את המימוש האפשרי בשפות החדשות (List, MyList).
- ומה עושים עם עץ ריק?

ועוד קצת הקדמה...

- הנקודות שאעלה הן לאחר התנסות בהוראה של כ- 170 מורים בהשתלמויות במהלך השנה האחרונה.
- אתמקד במעבר מ- Pascal ל- Java, אם כי הדברים מאד דומים גם ביחס ל C ו- C#.
- אני **לא מלמדת**... אלא **רק מצביעה** על המקומות החשובים ומתארת את **ההמלצות שלי להוראה**.

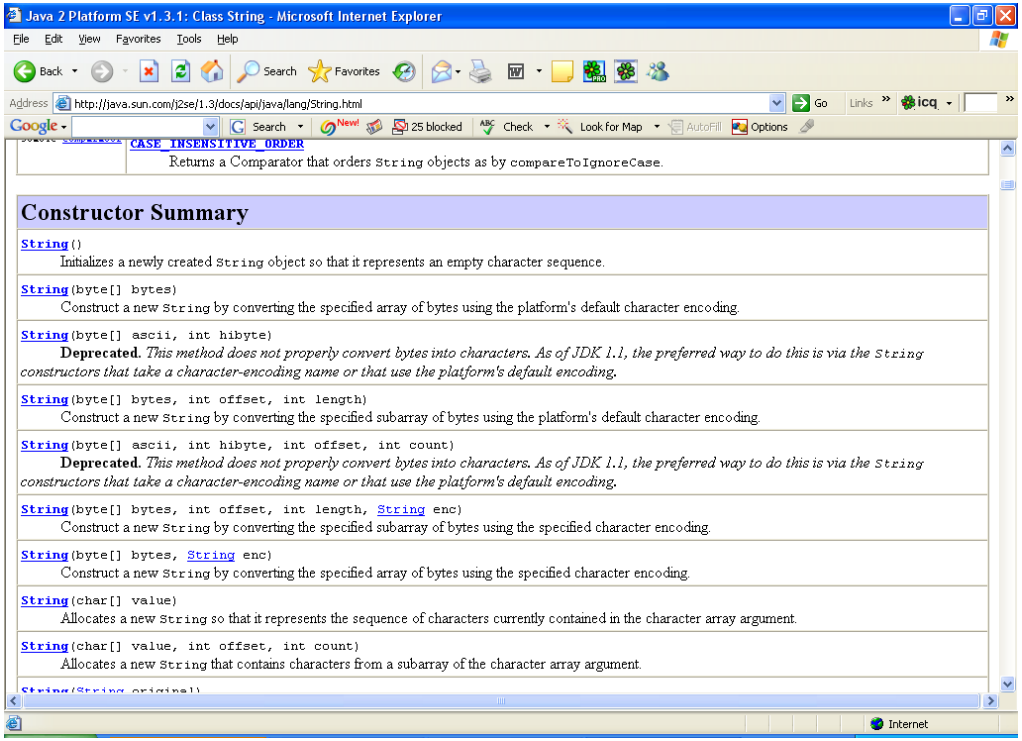
מחרוזות (1)

- הטיפוס String הוא טיפוס בנוי בשפה.



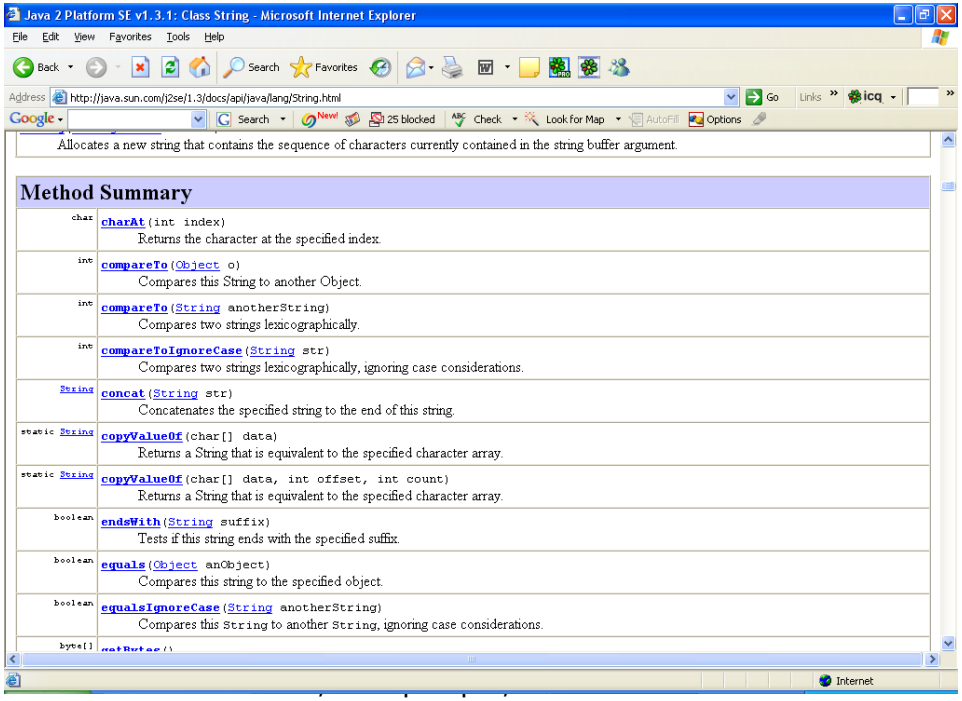
מחרוזות (2)

- ערכים של משתנה מהטיפוס String הם עצמים.
- בשפה אף מוגדרות פעולות בונות עבור עצמים מן הטיפוס String, למרות שלא חייבים להשתמש בהן.



מחרוזות (3)

- על עצמים מן הטיפוס String ניתן להפעיל פעולות המוגדרות בטיפוס.
- למשל: indexOf, substring, length, replace



מחרוזות (4) - ומה המשמעויות של כל זה?

- מאחר וערך של משתנה מטיפוס String הוא עצם, אזי למעשה הערך איננו העצם אלא הוא המצביע אל העצם.
  בהתאם לכך הפלט של קטע התכנית הבא יהיה **No**

```
String s1 = "abcde";
String s2 = "abc";
s2 = s2 + "de";
if  (s1 == s2)
        IO.writeln("Yes");
else
        IO.writeln("No");
```

מחרוזות (5) - ומה המשמעויות של כל זה?

- אם רוצים להתייחס אל השוואה לקסיקוגראפית כדי לקבל פלט **Yes** - יש להשתמש בפעולה equals המוגדרת בטיפוס String.

```
String s1 = “abcde”;
String s2 = “abc” + “de”;
if  (s1.equals(s2))
        IO.writeln(“Yes”);
else
        IO.writeln(“No”);
```

מחרוזות (6) - ומה המשמעויות של כל זה?

- **אבל**... פלט של קטע התכנית הבא יהיה דווקא **Yes** .

```
String s1 = “abcde”;
String s2 = “abcde”;
if  (s1 == s2)
        IO.writeln(“Yes”);
else
        IO.writeln(“No”);
```

- וזה בגלל מנגנון שמירת הקבועים המחרוזתיים בשפה "החוסך" בזיכרון לאחסון הקבועים, ואי לכך בהשמה השנייה משים מצביע אל אותה מחרוזת במאגר הקבועים.

מחרוזות (7) - ומה עם פעולות?

- מאחר ומחרוזות הן עצמים, כאשר מפעילים עליהן פעולות המוגדרות בטיפוס String, משתמשים בתחביר הפעלת פעולה על עצם. כלומר: על העצם ... הפעל את הפעולה ...

```
String s1 = “abcde”;
int p = s1.indexOf(‘d’);
```

מחרוזות (8) - ומה עם פעולות?

- אבל מה קורה כאשר נרצה להגדיר פעולה משלנו שתפעל על מחרוזות?, למשל: פעולה הפועלת על מחרוזת ומחזירה את מספר המופעים של תו מסוים בתוך מחרוזת.

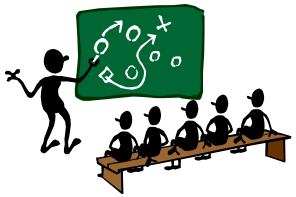
- מאחר והטיפוס String הוא טיפוס בנוי בשפה, איננו יכולים להרחיב את אוסף הפעולות שלו. המשמעות היא:
  - עלינו להגדיר את הפעולה **כפעולה סטטית** (static).
  - הפעולה תקבל את **המחרוזת כפרמטר**.
  - זימון הפעולה **לא** יהיה בנוסח: על העצם ... הפעל את הפעולה ...

מחרוזות (9) - ומה עם פעולות?

- למשל:

```
public class ExString
{
	public static int numOfChar(String s, char c)
	{
		int counter = 0;
		for (int i=0; i < s.length(); i++)
			if  ( s.charAt(i)  ==  c )
				counter ++;
		return counter;
	}

	public static void main(String[] args)
	{
		String s1 = "abcabcaasa";
		int m = numOfChar(s1, 'a');
		IO.writeln(m);
	}
}
```



מחרוזות - ומה עם ההוראה?

- **באשר ל:** פעולות בטיפוס המוגדר מראש
  - **לא קרה כלום**, גם קודם היו לנו פעולות מוגדרות מראש עבור מחרוזות.
- **באשר ל:** השוואת מחרוזות
  - **לא נורא**, יקפידו על שימוש ב equals במקום ב ==
- **באשר ל:** דרך הפעלת הפעולות על עצמים
  - **קצת מבלבל**, עד עכשיו השתמשו בזימון פעולות "רגיל" ועכשיו יצטרכו להתרגל לזימון פעולות "על העצם..."
- **באשר ל:** הגדרה והפעלה של פעולות "שלנו" על עצמים
  - **זה כבר... מבלבל ו... מאד!!!**, עד שהתרגלנו להפעלת פעולה על העצם מחרוזת ונמנענו מלהעביר אותה כפרמטר,

    פתאום כן צריך להעביר אותה כפרמטר וגם לזמן אותה "רגיל".

ממחרוזות .... למערכים

מערכים (1)

- ערך של משתנה מטיפוס מערך אף הוא עצמים.
- כלומר, ערך של משתנה מטיפוס מערך הוא מצביע.
- הקצאת הזיכרון למערך מתבצעת למעשה בשני שלבים עיקריים:
  - הקצאת מצביע אל המערך.
  - הקצאת תאי המערך.

```
int[] a1;
a1 = new int[10];
```

או

```
int[] a1 = new int[10];
```

מערכים (2)

- המשמעות למשל:

```
int[] a1 = new int[10];
for (int i=0; i<10; i++)
    a1[i] = i;
int[] a2 = a1;
for (int i=0; i<10; i++)
    IO.writeln("a1:" + a1[i] + " a2:" + a2[i]);
a2[5] = 555;
for (int i=0; i<10; i++)
    IO.writeln("a1:" + a1[i] + " a2:" + a2[i]);
```

השמה של המצביע אל כל המערך

לכן כמובן שינוי עבור "שני" המערכים

מערכים (3)

- וגם אם נעשה כך, התוצאה תהיה בדיוק אותו דבר:

```
int[] a1 = new int[10];
for (int i=0; i<10; i++)
     a1[i] = i;
int[] a2 = new int[10];
for (int i=0; i<10; i++)
     a2[i] = i*100;
a2 = a1;
for (int i=0; i<10; i++)
     IO.writeln("a1:" + a1[i] + " a2:" + a2[i]);
```

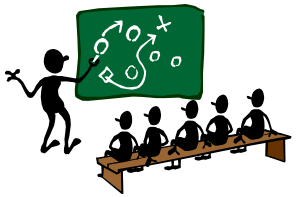
הקצאה נפרדת והשמת ערכים

מערכים (4)

- מה **שאגב** אפשרי בפסקל, והיה מביא **לתוצאה שונה**:

```
var
  a1, a2 : array[0..9] of integer;   i: integer;
begin
    for i:=0 to 9 do
       a1[i] := i;
    a2 := a1;
    for i:=0 to 9 do
       writeln('a1: ' , a1[i] , '  a2: ' , a2[i]);
    a2[5] := 555;
    for i:=0 to 9 do
       writeln('a1: ' , a1[i] , '  a2: ' , a2[i]);
end.
```

השינוי חל רק על המערך a2



מערכים - ומה עם ההוראה?

- **באשר ל:** הקצאת תאי המערך
  - **לא קרה כלום**, כך צריך להגדיר... וזהו.
- **באשר ל:** היות משתנה המערך - מצביע
  - **לשים לב**, לא לעשות השמה של מערך אל מערך, כי כך אובד הקשר לכל חלקי הזיכרון שהוקצו למערך אליו משימים.

    הזיכרון עצמו לא מטריד אותנו..., אבל אין בזה הגיון.
  - **טיפ**, אם רוצים להעתיק מערך, יש פעולה מתאימה של המערכת, עבור שני מערכים שהוצהרו והוקצו התאים שלהם:

    ```
    System.arraycopy(a1,0,a2,0,10);
    ```

    קרי: העתק מן המערך a1 החל מן המקום 0, אל המערך a2 החל מן המקום 0, 10 תאים בהתאמה. כל הערכים הנומריים יכולים להיות מותאמים כנדרש.
- **באשר ל:** פרמטר כמערך
  - **משנה זהירות**, הסבר בסעיף הבא

ממערכים .... לפרמטרים

העברת פרמטרים

- כל העברת הפרמטרים ב- java היא על פי:

פרמטר של ערך

כלומר

הקצאת זכרון מקומית עבור הפרמטרים

- עבור פרמטרים מן הטיפוסים הבסיסיים (int, float, double, char, boolean)
  - מועברים הערכים אל הפרמטרים, וכל שינוי שעובר עליהם לא משפיע על המשתנים "שנשלחו".
- מה קורה במחרוזות? – אותו הדבר
  - למרות שכאן הערך הוא המצביע אל העצם – עדיין מועבר ערך, ולכן גם כאן כל שינוי שעובר על הפרמטר בפעולה לא משפיע על המשתנה ששלח את הערך.
- מה קורה במערכים? – גם אותו הדבר – ממש לפי הכלל
  - מועבר המצביע אל המערך ולכן - כל שינוי שחל על תאי המערך, חל על אותו מקום בזיכרון!!!

העברת פרמטרים – מחרוזות (דוגמה 1)

- **נסתכל על הפעולה הבאה:**

```
public static void deleteAnySecondChar1(String s)
{     for (int i=0; i < s.length(); i++)
            s = s.substring(0, i) + s.substring(i+2);
      IO.writeln("in method: " + s);
}
```

- **עבור הזימון הבא:**

```
String s1 = "abababababa";
deleteAnySecondChar1(s1);
IO.writeln(s1);
```

- **יתקבל הפלט:**

```
in method: aba
abababababa
```

העברת פרמטרים – מחרוזות (דוגמה 2)

- **נסתכל על הפעולה הבאה:**

```
public static String deleteAnySecondChar2(String s)
{       for (int i=0; i < s.length(); i++)
            s = s.substring(0, i) + s.substring(i+2);
        return s;
 }
```

- **עבור הזימון הבא:**

```
String s1 = "abababababa";
String s2 = deleteAnySecondChar2(s1);
IO.writeln(s1);
IO.writeln(s2);
```

- **יתקבל הפלט:**

```
abababababa
aba
```

העברת פרמטרים – מערכים (דוגמה 1)

- **נסתכל על הפעולה הבאה:**

```
public static void zeroAnySecondCell1(int[] a)
{
        for (int i=1; i < a.length; i+=2)
                a[i] = 0;
        IO.write("in method: ");
        for (int i=0; i<a.length; i++)
                IO.write(a[i] + "    ");
        IO.writeln();
 }
```

העברת פרמטרים – מערכים (דוגמה 1 - המשך)

- **עבור הזימון הבא:**

```
int[] a1 = new int[10];
for (int i=0; i<10; i++)
      a1[i] = i;
zeroAnySecondCell1(a1);
for (int i=0; i<a1.length; i++)
       IO.write(a1[i] + "    ");
IO.writeln();
```

- **יתקבל הפלט:**

```
in method: 0    0    2    0    4    0    6    0    8    0
0    0    2    0    4    0    6    0    8    0
```

העברת פרמטרים – מערכים (דוגמה 2)

- **נסתכל על הפעולה הבאה:**

```
public static int[] zeroAnySecondCell2(int[] a)
{
        for (int i=1; i < a.length; i+=2)
             a[i] = 0;
        return a;
 }
```

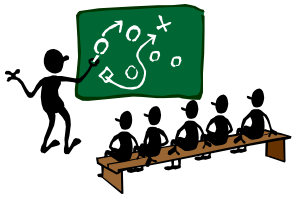
העברת פרמטרים – מערכים (דוגמה 2 - המשך)

- **עבור הזימון הבא:**

```
int[] a2 = new int[10];
for (int i=0; i<10; i++)
      a2[i] = i;
a2 = zeroAnySecondCell2(a2);
for (int i=0; i<a2.length; i++)
     IO.write(a2[i] + "    ");
IO.writeln();
```

- **יתקבל הפלט:**

```
0   0   2   0   4   0   6   0   8   0
```



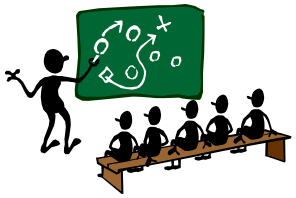
פרמטרים - ומה עם ההוראה? (1)

- **באשר ל:** טיפוסים "רגילים"
  - **לא קרה כלום**, כרגיל – מועבר הערך.

- **באשר ל:** מחרוזות
  - **לא נורא**, למרות שהן "אחרות" – מתנהגות כמו טיפוסים "רגילים".
  - **המלצה להוראה:** להתרגל שעבור ניסוח של "כתוב פעולה אשר ... ***עושה משהו למחרוזת*** ...", כדאי להגדיר פעולה המחזירה את המחרוזת המעודכנת, ולהתרגל בזימון להשים את הערך המוחזר ישר לאותה המחרוזת.

כדוגמת:

```
s = deleteAnySecondChar2(s);
```



פרמטרים - ומה עם ההוראה? (2)

- **באשר ל:** מערכים
  - לשים לב, כל שינוי שחל בפעולה חל על המערך עצמו.
  - המלצה להוראה 1: בדומה למחרוזות אם מגדירים פעולה ש "צריכה לעשות משהו למערך" (מיון), כדאי להגדיר פעולה המחזירה את המערך המעודכן, ולהתרגל בזימון להשים את הערך המוחזר ישר לאותו המערך. – למרות שבכל מקרה הוא מתעדכן!!!

    כדוגמת: `a = zeroAnySecondCell2(a);`

    כך גם נשמור על אחידות בגישת ההוראה.
  - המלצה להוראה 2: אם מגדירים פעולה שהעיבוד שלה דורש שינוי של המערך, אך לא רוצים ש "מערך המקור" ישתנה, אזי מתאים להגדיר מערך נוסף מקומי לפעולה, להעתיק אליו את הערכים מן המערך שהתקבל כפרמטר, ולעשות על העותק את כל מה שרוצים.

מפרמטרים .... לרשומות

רשומות (1)

- המונח רשומה לא קיים.
- רשומה היא למעשה טיפוס (ממומש במחלקה).
- ואם אנו מדברים על מחלקה, גם אם נגדיר בה רק תכונות, אז אנחנו כבר מדברים על עצמים, ועל בניית עצמים.
- אחרי שיהיו לנו עצמים נוכל לפנות אל "שדות" שלהם – למעשה תכונות.
- התבנית הכי בסיסית לפנייה ישירה לתכונה היא תוך שימוש ב '.' , קרי: פנה אל התכונה ... של העצם ...

רשומות (דוגמה 1)

- **עבור הטיפוס** Student:

```
public class Student
{       String name;
        int average;
 }
```

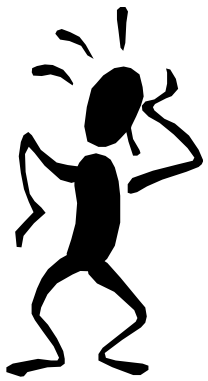
- **תוגדר התכנית** StudentProg:

```
public class StudentProg
{    public static void main(String args)
     {      Student stu1 = new Student();
            stu1.name = IO.readString();
            stu1.average = IO.readInt();
     }
}
```

רשומות (2)

- אפשר מבחינת השפה להגדיר טיפוס (שאמור להיות רשומה...) כמחלקה פנימית – מבנה שנראה אולי כקרוב לרשומה. אך עדיין אנחנו מדברים על טיפוס ועל עצמים.

- לדעתי הגדרה של מחלקה פנימית למחלקה, תגרום בלבול גדול לתלמידים.



רשומות (דוגמה 2)

- **למשל התכנית** StudentProg:

```
public class StudentProg
{     public static class Student
      {       String name;
              int average;
      }

      public static void main(String[] args)
      {       Student stu1 = new Student();
              stu1.name = IO.readString();
              stu1.average = IO.readInt();
      }
}
```

רשומות (3)

- מעבר לכך השימוש המשמעותי ברשומות לצורך פתרון בעיות הוא בדרך כלל עם מערך של רשומות.

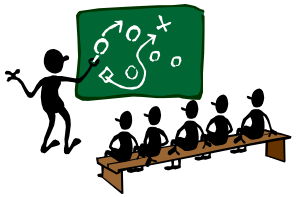
במקרה שלנו - מערך של עצמים.

הבנה ועבודה עם מערך של עצמים אינה פשוטה להערכתי לתלמידי יסודות, הלומדים בגישה הפרוצדוראלית.

רשומות (דוגמה 3)

- **למשל התכנית** StudentProg:

```
public class StudentProg
{     public static class Student
      {       private String name;
              private int average;
       }
      public static void main(String[] args)
      {       Student[]  stu = new Student[30];
              for (int i=0; i<stu.length; i++)
              {       stu[i] = new Student();
                      stu[i].name = IO.readString();
                      stu[i].average = IO.readInt();
               }
      }
}
```



רשומות - ומה עם ההוראה? (1)

- **האמת ???:** לא ללמד
- **למה ???:** לא יקרה כלום
- **מה הבעיות לדעתי ???:**
  - תכנות מונחה עצמים – הוחלט כפי הנראה לכלול רק ביחידה "עיצוב תוכנה". לכן ללמד משהו דמוי "חצי הריון" ממש לא מומלץ.
  - אם **כן** תכנות מונחה עצמים – **אז כמו שצריך**

כלומר: תכונות private, פניה לתכונות רק דרך פעולות קובעות ומאחזרות, טיפוס עצמאי ולא פנימי (סטטי), הסבר ראוי על פעולות בונות ולא הבלעה ללא הבנה של שימוש בפעולה בונה ברירת מחדל.

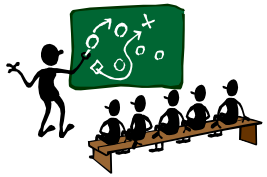
אחרת נעשה נזקים לקראת השלב בו כן נלמד תכנות מונחה עצמים - וחבל !!!

מרשומות ... ליחידת ספרייה

יחידת ספריה

- מחלקה ובה פעולות "רגילות" לשימוש.
- הפעולות יוגדרו כ static.
- הפעולות יזומנו תוך שימוש בשם המחלקה.
- בגדול – הזימון הוא בדיוק כמו בפעולות שהתלמידים כבר הגדירו בתוך מחלקה, יש רק תוספת של שם הטיפוס.
- **יחידת ספריה - ומה עם ההוראה?**
  - אני לא רואה שום בעיה.
  - בפרט מאחר והתלמידים כבר יהיו רגילים לשימוש כזה:
    - מן הפונקציות המתמטיות: Math.sqrt(n)
    - משימוש בספריית קלט/פלט כדוגמת IO, למשל:

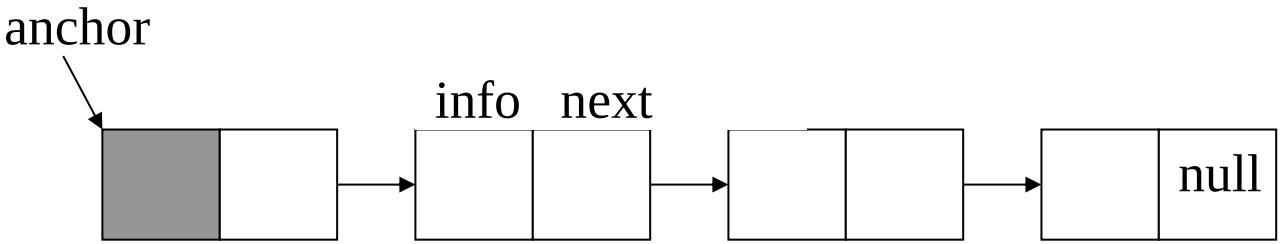
```
IO.writeln("Hag Sameach");
```



מיחידת ספרייה .... לרשימה

רשימה (1)

- הטיפוס רשימה נכלל ביחידת הלימוד "עיצוב תוכנה".
- המימוש בחומרי הלמידה החדשים שמפותחים הוא מונחה עצמים – כראוי לדעתי.
- לא אתייחס כאן אל המימוש עצמו, אלא רק לסוגיית סיווג הפעולות המופעלות על רשימה.



רשימה (2)

- תזכורת לצורת העבודה שלנו עד כה:
  - הייתה קיימת יחידת ספרייה מוסכמת על כולם

    (אפילו אני משוכנעת שכולנו משתמשים ביחידה list2.pas).
  - כאשר הוגדרו פעולות נוספות על רשימה הן יכלו להיות מוגדרות באחת משלושת האופנים הבאים:
    - נכתבו **כפעולות בתוך תכנית** אשר משתמשת ביחידת הספרייה.
    - נכתבו **בתוך יחידת ספרייה נוספת** שנקרא לה למשל: myList,

      כך שהתכנית השתמשה בשתי היחידות: uses list, myList;
    - נכתבו **בתוך היחידה list** בנוסף לפעולות "המוסכמות" הכתובות בה.

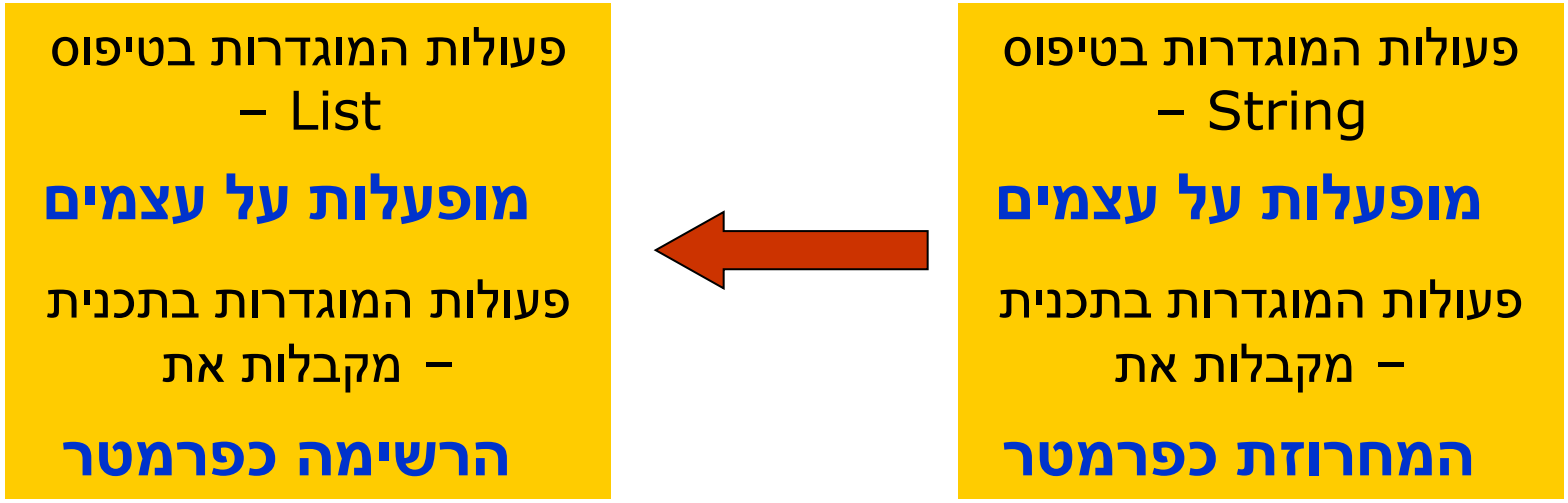
      בשיטה זו אני מניחה שמעטים השתמשו מתוך חשש להערכה בבחינת הבגרות.
      - מה שמותר לשימוש בבחינת הבגרות בפנים.
      - מה שאסור לשימוש בבחינת הבגרות בחוץ.

רשימה (3)

- אז מה נעשה עם הגישה השנייה ? :

"הפעולות נכתבו בתוך התכנית אשר משתמשת ביחידת הספרייה"

- אנחנו בבעיה! - בדיוק אותה הבעיה שהועלתה באריכות בדיון בטיפוס String



רשימה (4)

- אז מה נעשה עם הגישה הראשונה ? :

" נכתבו בתוך יחידת ספרייה נוספת שנקרא לה למשל: MyList"

- אנחנו בבעיה **יותר חמורה**! - אם יש לנו שני טיפוסים, אז מי העצם שלנו? הוא מטיפוס List או מטיפוס MyList.

אם העצם מטיפוס List

אפשר להפעיל עליו

פעולות המוגדרות בטיפוס List

אם העצם מטיפוס MyList

אפשר להפעיל עליו

פעולות המוגדרות בטיפוס MyList

רשימה (5)

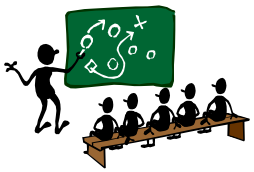
- אז מה נעשה ???
- בשפות התכנות יש מה לעשות !!!

אם נגדיר את הטיפוס MyList **כיורש** מן הטיפוס List.

- **המשמעות:**
  - העצמים יהיו מהטיפוס MyList.
  - אפשר יהיה להפעיל עליהם את כל הפעולות המוגדרות ב List וגם ב MyList.
  - סגנון הפעלת הפעולות הוא על עצמים (ולא העברת הרשימה כפרמטר).

רשימה (6)

- **כלומר – אפשר להרוויח את הכל :**
  - כל הפעולות הן פעולות המופעלות על עצמים.
  - יש הפרדה בין פעולות "מוסכמות" לפעולות אחרות.



- **אז – למה לא? :**
  - כי הורשה לא נכללת בתכנית הלימודים.
  - נושא ההורשה **אמנם** נחשב כמתקדם ומורכב, וגם כך יש לנו הרבה תכנים מתקדמים ביחידה זו.
- **אני נוטה –** בכל זאת לשקול גישת פתרון זו, מבלי להעמיק – בעיקר מאחר ואני רואה הרבה בעיות עם הפתרונות האחרים.
- **לפי טיוטות ראשונות** של החומרים המפותחים באוניברסיטה העברית – תהיה הבחנה בין פעולות "פנימיות" לטיפוס לפעולות "חיצוניות" לטיפוס.

מרשימה .... לעץ

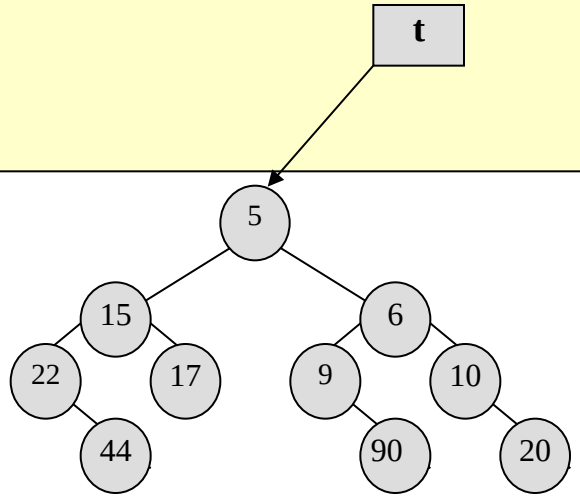
ומה קורה עם העץ ? (1)

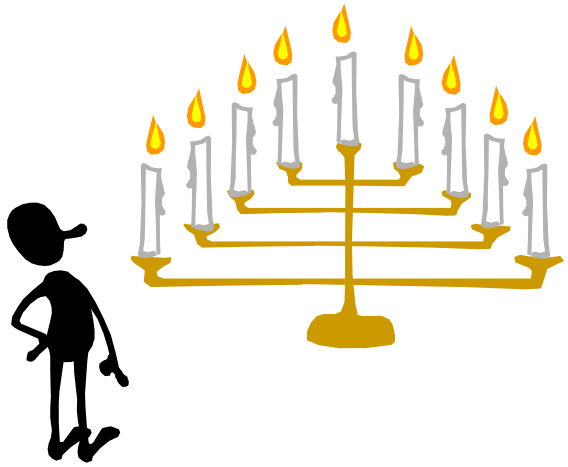
- כאן הבעיה היא אובייקטיבית של הפרדיגמה.
- מוקד הבעיה הוא בפעולה האם-עץ-ריק?.
- עץ ריק הוא עץ שגם אין לו שורש, כלומר הערך של המצביע אל השורש שלו הוא ריק (null).
- כלומר, בתכנות מונחה עצמים – עצם שעוד לא נבנה – עצם שהוצהר אך לא נעשה עבורו new.
- אם כך, אם עוד אין עצם, אז איך נשאל אם הוא ריק?
- כלומר, לפחות פעולה זו **לא יכולה** להיות פעולה המופעלת על עצם מטיפוס עץ. העצם לא יכול מחד לא להיות קיים, ומאידך להפעיל עליו פעולה שתבדוק אם הוא ריק.
- קשה למצוא פתרון אלגנטי לסוגיה זו, שיתיישב עם הייצוג של הטיפוס, עם כלי התכנות, ועם האלגוריתמים שאנו מגדירים עבור עץ.

ומה קורה עם העץ ? (2)

- **למשל הפעולה תוגדר:**

```
public static boolean isEmpty(Tree t)
{
        return ( t == null );
}
```



בהצלחה לכולנו

תודה וחג שמח