# פרויקט , מחלקה

- הקבצים בג'אווה מאוגדים בפרויקטים. כל **פרויקט** מכיל מספר מחלקות.
- **בחלק מהמחלקות מוגדרות תכונות** של עצם מטיפוס המחלקה ומוגדרות **הפעולות** שניתן לבצע עליו ועימו (כמו למשל המחלקה Bucket.java).
- **בחלק מהמחלקות מוגדרת פעולה ראשית (main)**. הפעולה הראשית מופעלת אוטומטית בעת הרצת המחלקה. במחלקות אלה יכולות להיות מוגדרות גם **פעולות נוספות**.

# יצירת עצמים וביצוע פעולות על עצמים

**עצם מורכב מתכונות** שמתארות את המצב של העצם ושניתן לשנות אותן, **ומפעולות** שניתן לבצע עליו.

**הוראה ליצירת עצם חדש והצבת הפנייה אליו בתוך משתנה:**

| |
|---|
| שם המחלקה שם המשתנה = **new** שם המחלקה (ערכים) |

למשל, ההוראה הבאה תיצור עצם חדש מטיפוס Bucket (דלי) ותציב הפנייה אליו בתוך משתנה b1 :

```
Bucket b1 = new Bucket (12);
```

**פעולות הפועלות על עצמים:**

פעולה לבניית עצם, פעולות מאחזרות (מחזירות את הערכים של התכונות של העצם), ופעולות נוספות.

**הפעלה של פעולה:** (ערכים) שם הפעולה.משתנה שיש בו הפניה לעצם  למשל, ;(9) b1.fill

יתכנו פעולות שלשם הפעלתן צריך לתת **מספר ערכים**. הפעלה של פעולה כזו:

(ערךn, .... ערך2, ערך1) שם הפעולה.משתנה שיש בו הפניה לעצם

ויתכנו פעולות **שלא מחכות לשום ערך**. הפעלה של פעולה כזו: () שם הפעולה.משתנה שיש בו הפניה לעצם

- כמו בפעולות שהכרנו עד כה, גם כאשר פעולה שפועלת על עצם מחזירה ערך, צריך לעשות משהו עם הערך המוחזר!
- ניתן לבצע הרכבה של פעולות. למשל, משמעות הביטוי: ;(()b2.fill(b1.getCurrentAmount היא הפעלה של הפעולה fill על העצם שההפניה אליו נמצאת במשתנה b2. הערך שהפעולה תקבל, יהיה הערך המוחזר מהפעולה getCurrentAmount המופעלת על עצם שההפניה אליו נמצאת במשתנה b1.


© כל הזכויות שמורות, תמר פז

# מחלקות

**מבנה של מחלקה שמוגדרות בה תכונות של עצם מטיפוס המחלקה, ופעולות שניתן לבצע עליו ועימו**

**public class** *שם המחלקה*

**{**

**// הגדרת התכונות של המחלקה**

private טיפוס תכונה1 שם תכונה1 ;

private טיפוס תכונה2 שם תכונה2 ;

……………………………………….

private טיפוס תכונהn שם תכונהn ;

**private** = התכונות הן משתנים פנימיים וניתן להשתמש בהן רק בקובץ הנוכחי שמגדיר את המחלקה. התכונות לא מוכרות למי שמשתמש במחלקה.

**// הגדרת הפעולה הבונה -- הפעולה הבונה מחזירה עצם מטיפוס המחלקה**

**public** ***שם המחלקה (טיפוס ערך, טיפוס ערך,......, טיפוס ערך)***

{

this.תכונה1 = ערך;

this.תכונה2 = ערך;

……………………..

this.תכונהn = ערך;

}

**public** = ניתן להשתמש במחלקה זו ממחלקות אחרות

הפעולה הבונה דואגת לתת ערך לכל התכונות של העצם הנבנה. הערכים של התכונות יכולים להיות פרמטרים של הפעולה הבונה, יכולים להתקבל באמצעות הוראת קלט או להקבע על-ידי הפעולה הבונה

**// הגדרות של הפעולות המאחזרות והפעולות הנוספות**

**}**

**הגדרה של פעולה שאיננה הפעולה הבונה:**

**public** ***טיפוס הערך המוחזר שם הפעולה (טיפוס ערך, טיפוס ערך,......, טיפוס ערך)***

{
הוראות לביצוע
}

**הרשאות גישה:**

**public** = הרשאת גישה פומבית. ניתן לגשת לפעולה גם ממחלקות אחרות.

**private** = הרשאת גישה פרטית. ניתן לגשת לתכונה או לפעולה רק מקובץ המחלקה הנוכחי.

- חתימה של פעולה בונה שלא מקבלת ערכים: **public** ***שם המחלקה ()***
- חתימה של פעולה (שונה מהפעולה הבונה) שלא מקבלת ערכים:

***הרשאת גישה טיפוס הערך המוחזר שם הפעולה ()***

דוגמא: public int getDay( )

- כאשר פעולה לא מחזירה ערך, בחתימה של הפעולה, במקום הערך המוחזר, רושמים: void. למשל, public void fill (double amountToFill)

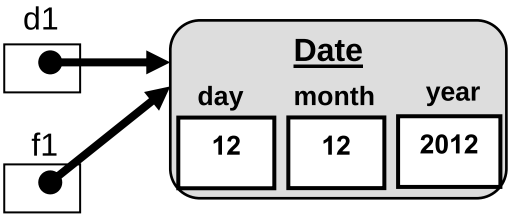

© כל הזכויות שמורות, תמר פז

- כדי להקל את ההתמצאות בקובץ, נפתח תמיד בפעולות המאחזרות (שמחזירות את הערכים של התכונות של העצם). לאחריהם יופיעו שאר הפעולות. הפעולה האחרונה תהייה הפעולה toString שיוצרת מחרוזת מהערכים של התכונות של העצם הנוכחי.
- בכל הפעולות, ההתיחסות לתכונה של העצם עליו מתבצעת הפעולה, היא באמצעות המילה **this**. המילה **this** מציינת את העצם הנוכחי עליו מתבצעת הפעולה.

- ג'אווה מאפשרת להגדיר מספר פעולות בעלות אותו שם בתנאי שהן שונות זו מזו ברשימת הפרמטרים. השינוי יכול להתבטא במספר הפרמטרים ו/ או בטיפוסים שלהם. בעת הזימון של הפעולה המהדר (הקומפיילר) יבצע את הפעולה המתאימה לרשימת הפרמטרים שתסופק לו.
לכן, אפשר למשל להגדיר שתי פעולות בונות באותה מחלקה. אחת שתקבל פרמטרים, ואחת שתקלוט מהמשתמש את הערכים הרצויים. החתימה של הפעולה הראשונה תהיה:
public Date (int day , int month , int year) והחתימה של הפעולה השניה תהיה:
public Date( ) .
המנגנון שמאפשר להגדיר מספר פעולות בעלות אותו שם נקרא **העמסה** (overloading).

- **בהעברת עצם כפרמטר, אנחנו מעבירים את ההפניה לעצם ולא את העצם עצמו!** כלומר, הפעולה מקבלת ממי שמזמן אותה הפניה לעצם ולא את העצם עצמו!
לכן, במחלקה שמשמאל, במשתנה d1 (של הפעולה הראשית) ובמשתנה f1 (של הפעולה test) נמצאת **הפניה** לאותו **עצם מטיפוס Date**. כתוצאה מכך, כאשר הפעולה test משנה את הערכים של התכונות של העצם שההפניה אליו נמצאת ב f1 , היא משנה גם את הערכים של התכונות של העצם שההפניה אליו נמצאת ב d1 (שהרי מדובר באותו עצם!). לכן, לאחר הפעלת הפעולה test, הערכים של התכונות של Date שההפניה אליו נמצאת ב d1, הם הערכים כפי שנקבעו בפעולה test

```
public class TestParemeter
{
    public static void main(String[] args)
    {
            Date d1 = new Date (10,9,2006);
            System.out.println ("before test d1 = " + d1.toString());
            test (d1);
            System.out.println ("after test d1 = " + d1.toString());
    }
    private static void test (Date f1)
   {
            f1.setDay (12);
            f1.setMonth (12);
            f1.setYear (2012);
          System.out.println ("in test f1 = " + f1.toString());
   }
}
```





© כל הזכויות שמורות, תמר פז

# מחלקות מוכנות

## המחלקה Math

- המחלקה Math היא **מחלקת שירות**. במחלקה כזו **לא מגדירים עצמים**.
- המחלקה Math מיובאת באופן אוטומטי ואין צורך לייבא אותה כדי להשתמש בה.
- הפעלה של פעולה במחלקת שירות: *שם המחלקה.שם הפעולה (פרמטרים)*

למשל, Math.abs (num);

**ממשק חלקי:**

| טיפוס הערך המוחזר | טיפוס הקלט | הערך המוחזר | הפעולה |
|---|---|---|---|
| מספר ממשי | מספר ממשי | הערך המוחלט של הפרמטר | **abs** |
| מספר שלם | מספר שלם | | |
| מספר ממשי | מספר ממשי | הערך השלם התחתון של הפרמטר | **floor** |
| מספר ממשי | שני מספרים ממשיים | הערך של הפרמטר הגדול יותר | **max** |
| מספר שלם | שני מספרים שלמים | | |
| מספר ממשי | שני מספרים ממשיים | הערך של הפרמטר הקטן יותר | **min** |
| מספר שלם | שני מספרים שלמים | | |
| מספר ממשי | שני מספרים ממשיים | הפרמטר הראשון בחזקת הפרמטר השני | **pow** |
| מספר שלם | מספר ממשי | ערך מעוגל של הפרמטר | **round** |
| מספר ממשי | מספר ממשי | השורש של הפרמטר | **sqrt** |
| מספר ממשי | ------------------- | מספר אקראי בתחום 0-1 (כולל 0, לא כולל 1). | **random** |


© כל הזכויות שמורות, תמר פז

# המחלקה Scanner (קלט)

- Scanner היא **מחלקה שמאפשרת לקלוט נתונים מהמשתמש.**
- המחלקה Scanner לא מיובאת באופן אוטומטי על-ידי סביבת העבודה.
- כדי להשתמש במחלקה שלא מיובאת באופן אוטומטי יש צורך לייבא אותה למחלקה הנוכחית.

**ייבוא של מחלקה:** ***שם המחלקה.מיקום המחלקה*** **import**

ייבוא של המחלקה Scanner :   import java.util.Scanner;

- יצירת עצם מטיפוס Scanner והשמת הפניה אליו במשתנה היא כמו יצירת עצם מטיפוס שהגדרנו

בסביבה:  Scanner ***שם המשתנה*** = new Scanner(System.in)

למשל, Scanner input = new Scanner(System.in);

- הפעלת פעולה על עצם מטיפוס Scanner היא כמו הפעלת פעולה על עצם מטיפוס שהגדרנו

בסביבה: ***שם העצם.שם הפעולה*** (*פרמטרים*)

למשל,  input.nextInt();

ייבוא של המחלקה Scanner. נכתב לפני הכותרת של המחלקה

```
import java.util.Scanner;
class Square
{
   public static void main(String[] args)
   {
      Scanner input = new Scanner(System.in)
       ……..
   }
}
```

יצירת עצם מטיפוס Scanner והצבת הפניה אליו במשתנה input .
אפשר לבחור שם משתנה כרצונכם.
צריך יהיה להשתמש בו במשך כל המחלקה

**הוראות קלט:**

**input.nextInt()** -- הוראה לקליטת מספר שלם.

**input.nextDouble()** -- הוראה לקליטת מספר ממשי.

**input.nextBoolean()** -- הוראה לקליטת ערך בוליאני.

© כל הזכויות שמורות, תמר פז

# המחלקה String

- String היא מחלקה מוכנה.
- עצם מטיפוס String הוא מחרוזת (או רצף של תווים).
- המחלקה String מיובאת באופן אוטומטי ואין צורך לייבא אותה כדי להשתמש בה.
- המחלקה String היא המחלקה היחידה בה לא משתמשים במילה השמורה new על מנת ליצור עצם!

למשל, הוראה ליצירת עצם מטיפוס String והצבת הפניה אליו בתוך משתנה:

באמצעות השמה: **String שם המשתנה = "רצף של תווים"**

או באמצעות קלט: **String שם המשתנה = input.next( );**

כמובן שניתן לפצל לשתי הוראות:

- הגדרת משתנה שיכול להכיל הפניה לעצם מטיפוס String: **String שם המשתנה;**

  למשל, String st1;
- הצבת הפניה לעצם מטיפוס String בתוך המשתנה

  באמצעות הוראת השמה: **שם המשתנה = "רצף של תווים";**

  או באמצעות הוראת קלט: **שם המשתנה = input.next ();**

- **פעולה לקליטת ערך למחרוזת:**

  א. אפשרות ראשונה (כאשר המחרוזת מסתיימת ב enter או ברווח או ב Tab):

*שם המחרוזת* = input.next()

  ב. אפשרות שניה (כאשר המחרוזת עשויה להכיל רווחים):

*שם המחרוזת* = input.nextLine()

- **פעולה לקליטת ערך לתו:**

אין הוראה מיוחדת לקליטת ערך מטיפוס char. משתמשים בהוראה next לקליטת מחרוזת ומפעילים עליה את הפעולה charAt(0) המחזירה את התו הראשון מהמחרוזת.

*שם המחרוזת* = input.next()charAt(0)

- פעולת השרשור **+** מאפשרת לבנות מחרוזת. למשל, st1= "hello " + name;
- לא ניתן לשנות ערך של עצם מטיפוס String. לכן, אף אחת מהפעולות במחלקה String לא משנה את העצם עליו היא פועלת! אבל מוגדרות במחלקה String מספר פעולות שמחזירות עצם מטיפוס String.


© כל הזכויות שמורות, תמר פז

- מממשק חלקי של המחלקה String התו הראשון נמצא במקום מספר 0.

| **תיאור הפעולה** | **כותרת הפעולה** |
|---|---|
| הפעולה הבונה: יוצרת עצם מטיפוס String | String (רצף של תווים) |
| מקבלת כפרמטר עצם מטיפוס String. מחזירה true אם הערך של המחרוזת הנוכחית שווה לערך של הפרמטר. ומחזירה false אחרת. | boolean equals (String otherString) |
| מקבלת כפרמטר עצם מטיפוס String. מחזירה מספר שלילי אם המחרוזת הנוכחית מופיעה לפני הפרמטר על-פי הסדר המילוני, מספר חיובי אם המחרוזת הנוכחית מופיעה אחרי המחרוזת על-פי הסדר המילוני, 0 אם המחרוזות זהות. | int compareTo (String otherString) |
| מחזירה את מספר התווים במחרוזת | int length() |
| מחזירה את התו שנמצא במקום index במחרוזת. | char charAt (int index) |
| מקבלת כפרמטר עצם מטיפוס String או משתנה מטיפוס char. מחזירה את המיקום הראשון בו מופיע הפרמטר בתוך המחרוזת הנוכחית. אם הפרמטר אינו מופיע, יוחזר 1-. | int indexOf (String otherString)<br>int indexOf (char ch) |
| מקבלת כפרמטר עצם מטיפוס String או משתנה מטיפוס char. מחזירה את המיקום האחרון בו מופיע הפרמטר בתוך המחרוזת הנוכחית. אם הפרמטר אינו מופיע, יוחזר 1-. | int lastIndexOf (String otherString)<br>int lastIndexOf (char ch) |
| מחזירה מחרוזת חדשה זהה למחרוזת הנוכחית עם השינוי הבא: בכל מקום שבמחרוזת הנוכחית הופיע oldChar, יופיע במחרוזת החדשה newChar | String replace (char oldChar, char newChat) |
| מחזירה מחרוזת חדשה זהה למחרוזת הנוכחית עם השינוי הבא: בכל מקום שבמחרוזת הנוכחית הופיע oldSt, יופיע במחרוזת החדשה newSt | String replace (String oldSt, String newSt) |
| מחזירה מחרוזת חדשה זהה למחרוזת הנוכחית עם השינוי הבא: כל התווים נכתבים באותיות קטנות. | String toLowerCase() |
| מחזירה מחרוזת חדשה זהה למחרוזת הנוכחית עם השינוי הבא: כל התווים נכתבים באותיות גדולות. | String toUpperCase() |
| מחזירה מחרוזת חדשה זהה למחרוזת הנוכחית ללא תווי רווח בהתחלת המחרוזת ובסופה. | String trim() |
| מחזירה מחרוזת חדשה הזהה לקטע המחרוזת הנוכחית המתחיל בתו שנמצא במיקום first ומסתיים בתו שנמצא לפני המיקום last | String substring (int first, int last) |
| מחזירה מחרוזת חדשה הזהה לקטע המחרוזת הנוכחית החל מהתו שנמצא במיקום first ועד לסוף המחרוזת. | String substring (int first) |

הערה: toUpperCase , toLowerCase שימושיים בקלט של מחרוזת, כדי לאפשר למשתמש להקליד אותיות קטנות / גדולות כאוות נפשו. למשל, **String name = (input.next()).toUpperCase();**

# המחלקה Array

- מחלקה מוכנה נוספת היא המחלקה Array.
- **מערך (Array) הוא עצם שמכיל מספר נתונים מאותו סוג.**
- גם המחלקה Array מיובאת באופן אוטומטי ואין צורך לייבא אותה כדי להשתמש בה.
- הוראה להגדרת משתנה שיכיל הפניה לעצם מטיפוס מערך: ***טיפוס נתונים [ ] שם המשתנה ;***

  למשל, char [ ] tavim;

  בשלב זה עדיין לא נקבע מספר הנתונים שהמערך יכיל!

- הוראה ליצירת עצם מטיפוס מערך והצבת הפניה אליו בתוך משתנה:

***שם המשתנה = new טיפוס נתונים [אורך המערך]***

למשל, tavim = new char [10];

- כמו ביצירת העצמים האחרים, ניתן לאחד את הגדרת המשתנה עם יצירת העצם החדש והצבת ההפניה אליו בתוך המשתנה:

***טיפוס נתונים [ ] שם המשתנה = new טיפוס נתונים [אורך המערך]***

למשל, int[ ] grades = new int[35];

- נתונים של מערך יכולים להיות מכל טיפוס נתונים המוכר בשפה.
- לאחר שיצרנו את המערך אי אפשר לשנות את הטיפוס של הנתונים שלו.
- מספר הנתונים שהמערך יכיל נקבע רק בעת הפעלת הפעולה הבונה, ואי אפשר לשנות אותו.
- לכל נתון (איבר) במערך יש מספר סידורי המציין את מיקומו במערך. המיקום של נתון (איבר) במערך נקרא מציין (אינדקס). המציין של הנתון הראשון הוא תמיד המספר 0, המציין של הנתון השני הוא המספר 1 וכן הלאה.
- פנייה לאיבר במערך: ***שם המערך[מציין האיבר]***

**העברת מערך כפרמטר:**

- הצהרה על מערך כפרמטר פורמלי: בחתימה של הפעולה: (***טיפוס המערך [ ] שם המערך***)

  למשל, private static void pelet (int [ ] a1)

- כמו בהעברת עצמים אחרים, גם בהעברת מערך כפרמטר, הפרמטר המועבר הוא הפניה למערך. **הפעולה מקבלת ממי שמזמן אותה הפניה לעצם מטיפוס מערך ולא את המערך עצמו!** ובכל ההתייחסויות לפרטר הפעולה משתמשת במערך שההפניה אליו נמצאת בפרמטר.


© כל הזכויות שמורות, תמר פז

**מערך שכל איבר בו מכיל הפניה לעצם:**

- איבר של מערך יכול להכיל הפניה לעצם!
- כאשר האיברים של המערך מכילים הפניה לעצם, טיפוס הנתונים הוא המחלקה. למשל, ההוראה **HighJump [ ] student = new HighJump [5]** מגדירה משתנה בשם student שיכיל הפניה לעצם מטיפוס מערך בעל 5 איברים. כל אחד מהאיברים יכיל הפניה לעצם מטיפוס HighJump .
- ההוראה HighJump [ ] student = new HighJump [5] מפעילה את הפעולה הבונה לבניית עצם מטיפוס מערך בעל 5 איברים שכל אחד מהם יכיל הפניה לעצם מטיפוס HighJump . **בשלב זו עוד לא נבנו** העצמים מטיפוס HighJump !!
- ההוראה **ar[i] = new HighJump ()** מפעילה את הפעולה הבונה לבניית עצם מטיפוס HighJump. ההוראה בונה **עצם יחיד** מטיפוס HighJump ומציבה הפניה אליו בתא מספר i במערך.

- הגישה לעצמים היא באמצעות תאי המערך: *שם המערך [מספר תא].פעולה (פרמטרים)*

```
public class StudenJump
 {
   public static void main(String[ ] args)
  {
     HighJump [ ] student = new HighJump [5];
     kelet (student);
     int numHigh = highest (student);
     System.out.println (student[numHigh].getName() + " achieved the highest average");
  }
  public static void kelet (HighJump [ ] ar)
  {
     for (int i=0; i<5; i++ )
        ar[i] = new HighJump ();
  }
  public static int highest (HighJump [ ] ar)
  { // פעולה שמקבלת את המערך ומחזירה את המיקום של בעל הקפיצה הממוצעת הגבוהה ביותר
     int numHigh = 0;
     for (int i=1; i< 5; i++ )
        if  (ar[i]. getAverageHigh() > ar[numHigh]. getAverageHigh() )
           numHigh= i;
     return numHigh;
  }
} // end of class
```


© כל הזכויות שמורות, תמר פז

# עצמים

- **עצם יכול להיות ערך המוחזר על-ידי פעולה ויכול להיות פרמטר של פעולה.**
- כאשר פעולה מקבלת כפרמטר עצם ששייך למחלקה הנוכחית היא מתייחסת לשני עצמים מטיפוס המחלקה. ההתיחסויות לעצם הנוכחי הן באמצעות הקידומת this. למשל this.currentAmount וההתיחסות לעצם שיתקבל כפרמטר היא באמצעות *שם העצם*. למשל, otherBucket.capacity.

  **ההתיחסות אל העצם שהתקבל כפרמטר איננה באמצעות this משום שהוא הפרמטר של המחלקה ולא העצם עליו פועלת המחלקה!**

- **עצם יכול להיות תכונה של מחלקה.**
- כאשר למחלקה יש תכונה מטיפוס עצם, ניתן להגדיר בה **פעולה בונה שמקבלת פרמטר מטיפוס העצם**, למשל, public StudentAge (String name, Date birthday)

  וניתן להגדיר **פעולה בונה שמקבלת פרמטרים מטיפוס התכונות של העצם.** למשל,

  public StudentAge (String name, int day , int month , int year)
- כאשר פעולה בונה מקבלת פרמטרים מטיפוס התכונות של העצם, צריך ליצור את העצם בתוך הפעולה הבונה. למשל,

```
public StudentAge (String name, int day , int month , int year)
 {
     this.name = name;
     this.birthday = new Date (day , month , year);
 }
```

- כאשר עצם משמש כתכונה, ניתן להרכיב פעולות משתי המחלקות: המחלקה אליה שייך העצם והמחלקה בה הוא משמש כתכונה. למשל, בביטוי st.getBirthday().getYear() מופעלת הפעולה getYear (מהמחלקה Date) על העצם המוחזר מהפעולה getBirthday (מהמחלקה studentAge). הדבר אפשרי כיוון שקודם מתבצעת הפעולה getBirthday המחזירה עצם מטיפוס Date, ולאחר מכן מופעלת הפעולה gerYear שמוגדרת במחלקה Date
- עצם שאחת מתכונותיו הוא עצם נקרא: **עצם מורכב.**
- גם במחלקה שאחת מהתכונות שלה היא מטיפוס עצם, ניתן להגדיר פעולה בונה מעתיקה.
- **מערך הוא עצם! לכן גם מערך יכול לשמש כתכונה של מחלקה!**

- כאשר עצם מועבר לפעולת ההדפסה, הפעולה toString נקראת באופן אוטומטי. לכן כדי להדפיס את הערך המוחזר מהפעולה toString אפשר לקצר ולכתוב:

**System.out.println (שם העצם);**

הפעולה toString נקראת באופן אוטומטי גם כאשר משלבים בהוראת ההדפסה מספר מחרוזות. למשל,

**System.out.println (t1 + "has sweet fruit")**


© כל הזכויות שמורות, תמר פז

## פעולה בונה מעתיקה

פעולה בונה שמקבלת כפרמטר עצם מטיפוס המחלקה ובונה עצם חדש מטיפוס המחלקה שהוא העתק של העצם שהתקבל כפרמטר

**מבנה של פעולה בונה מעתיקה:**

```
public שם המחלקה (שם המחלקה  משתנה)
{
  this.תכונה1 = משתנה.תכונה1;
   ...
  this.תכונהn = משתנה.תכונהn;
}
```

**זימון של פעולה בונה מעתיקה:**

שם המחלקה  משתנה-הפניה לעצם החדש = new  שם המחלקה  (משתנה- הפניה לעצם קיים)

## תכונות מופע ותכונות מחלקה

**תכונת מופע:** תכונה שנוצרת עבור כל עצם (או: מאפיינת מופע מסוים). לכל מופע של המחלקה יש עותק מקומי של התכונה והיא מאפיינת אותו.

**תכונת מחלקה:** תכונה שמאפיינת מחלקה ולא מופע מסוים שלה. בזיכרון המחשב קיים רק עותק אחד של התכונה ללא קשר למספר העצמים שנוצרו מהמחלקה. המילה static בהגדרת התכונה מציינת כי מדובר בתכונה שאינה קשורה לעצם מסוים. למשל, ;private static int counter=0

- ניתן לפנות אל תכונת מחלקה באמצעות עצם מטיפוס המחלקה (באמצעות מופע של המחלקה). כלומר, **this.**_**שם התכונה**_ ואפשר לפנות אל תכונת מחלקה באמצעות השם של המחלקה. כלומר, _**שם התכונה.שם המחלקה**_. הפניה באמצעות שם המחלקה עדיפה כיוון שהיא מדגישה זו תכונה של מחלקה.
- תכונת מחלקה קיימת **לפני יצירת העצם הראשון** מטיפוס המחלקה. ולכן ניתן לגשת אליה גם לפני יצירת העצם הראשון.
- אם מגדירים תכונת מחלקה כבעלת הרשאת גישה פרטית **(private)** למשל, ;private static int counter=0 אפשר לגשת אליה ולשנות אותה רק מתוך המחלקה בה היא מוגדרת.
- אם מגדירים תכונת מחלקה כבעלת הרשאת גישה פומבית **(public)** למשל, ;public static int counter=0 אפשר לגשת אליה גם ממחלקות שאינן המחלקה בה היא מוגדרת. אפשר לראות את ערכה ממחלקות אחרות ואפשר גם לשנות את ערכה ממחלקות אחרות.


© כל הזכויות שמורות, תמר פז

## מערך: התכונה length

- length היא **תכונת מופע פומבית** של המחלקה Array. הפניה אליה היא באמצעות עצם. כלומר, length. *שם משתנה* למשל כך: ar.length
- התכונה length שומרת את מספר האיברים של המערך.
- כל תכונות המופע (הקשורות לעצם מסוים) שפגשנו עד כה היו בעלות הרשאת גישה פרטית (לכן ניתן היה לגשת אליהן רק מהמחלקה בה הן הוגדרו). length היא תכונת מופע בעלת הרשאת גישה פומבית ולכן ניתן לגשת אליה ממחלקות שונות.

| |
|---|
| **לא להתבלבל: במחלקה String, length היא פעולה ולכן מזמנים אותה כך: sen.length()** |
| **ובמחלקה Array , length היא תכונה! ולכן מזמנים אותה כך: ar.length** |

## פעולות מופע ופעולות מחלקה

**פעולת מופע:** פעולה שמתבצעת על מופע מסוים של המחלקה (על עצם מסוים).

**פעולת מחלקה**: פעולה שהמחלקה מבצעת ללא קשר למופע מסוים שלה. פעולה שמתבצעת על-ידי המחלקה ולא על עצם מסוים.

- המילה static בחתימה של פעולת מחלקה מציינת כי זו פעולה שמתבצעת על-ידי המחלקה ולא על עצם מסוים. למשל, public static int freePlace ()
- מכיוון שפעולת מחלקה מתבצעת על-ידי מחלקה ולא על-ידי עצם מסוים, ניתן לבצע אותה גם לפני שהוגדרו עצמים מהמחלקה.
- כיוון ש**פעולות מחלקה** אינן קשורות לעצם מסוים וניתן אפילו להפעיל אותן לפני שהוגדרו כלל עצמים מהמחלקה, פעולות מחלקה לא יכולות להשתמש בתכונות של עצם מסוים! הן **יכולות להשתמש רק בתכונות של המחלקה** (תכונות מחלקה בעלות הרשאה static).
- כאשר **תכונת מחלקה** היא בעלת הרשאת גישה פרטית (**private**) **לא ניתן לגשת** אליה ממחלקות אחרות. כדי שאפשר היה לדעת מהו ערכה גם במחלקות אחרות, יש לכתוב **פעולת מחלקה** (פעולה **static**) שמחזירה את ערכה של התכונה.


© כל הזכויות שמורות, תמר פז

## עצם מורכב שאחת מתכונותיו היא מטיפוס עצמו

עצם מורכב הוא עצם שבו עצם אחר משמש כתכונה, ובפרט העצם שמשמש כתכונה יכול גם להיות מטיפוס המחלקה של העצם המורכב. במילים אחרות, עצם מורכב יכול גם להיות עצם שאחת מתכונותיו היא הפניה לעצם מאותו הטיפוס! למשל, לעצם מטיפוס המחלקה Bead יש שתי תכונות: צבע החרוז מטיפוס String והפניה לחרוז המושחל אחריו בשרשרת. ההפניה היא מטיפוס כתובת של חרוז.

```
public class Bead
{
        private String color;
        private Bead nextBead;

        public Bead (String color)
        {
                this.color = color;
                this.nextBead = null;
        }

        public Bead (String color, Bead nextBead)
        {
                this.color = color;
                this.nextBead = nextBead;
        }

        פעולות נוספות
}
```

- למחלקה Bead יש שתי פעולות בונות.
  פעולה בונה אחת מקבלת רק את הצבע של החרוז, ומציבה בתכונה nextBead את הערך null.

  **null הוא הפניה ריקה שאינה מובילה לשום עצם.** במילים אחרות, הפעולה הבונה דואגת שבתכונה nextBead לא תהיה הפניה לשום עצם.
- הפעולה הבונה השניה מקבלת צבע של חרוז והפניה לעצם מטיפוס Bead (חרוז).

## המחלקה הגנרית <Node <T

| Node**<T>** |
|---|
| **T** value<br>Node**<T>** next |
| Node(**T** x)<br>Node(**T** x, Node**<T>** next)<br>**T** getValue()<br>Node**<T>** getNext()<br>void setValue (**T** x)<br>void setNext (Node**<T>** next)<br>String toString() |

- עצם מטיפוס המחלקה Node נקרא **חוליה**. כיוון שאפשר לבנות מספר עצמים שבתכונה next של כל אחד מהם תהיה הפניה לעצם נוסף מטיפוס Node וליצור בכך **שרשרת חוליות**.
- הערך השמור בתכונה value הוא **הערך של החוליה**, והערך השמור בתכונה next הוא הפניה **לחוליה העוקבת** (או null במידה ואין חוליה עוקבת).
- **T** - הטיפוס של התכונה value נקבע רק בעת יצירת עצם מטיפוס המחלקה (יצירת מופע של המחלקה).
- במקום טיפוס הנתונים רושמים **T**, ובכל מקום שרוצים לציין את השם של המחלקה רושמים **<Node<T**. מסיבה זו, התכונה value היא מטיפוס **T**, ואילו התכונה next, שערכה יהיה הפניה לעצם (חוליה) מטיפוס המחלקה, היא מטיפוס **<Node<T** .

© כל הזכויות שמורות, תמר פז

- **השם של המחלקה** הוא Node, **והכותרת של המחלקה** היא: public class Node<T>
- במחלקה גנרית, ערך של חוליה (ערך של התכונה value) מוכרח להיות הפניה לעצם (למופע של מחלקה) ולא טיפוס פשוט! כלומר, הוא אינו יכול להיות char , double , int . לכן, בתוכנת ג'אווה לכל טיפוס פשוט מוגדרת מחלקה מקבילה: Integer עבור int , Double עבור double , Character עבור char, וכן הלאה. מחלקות אלה נקראות: **מחלקות עוטפות** (type wrapper classes). אם נרצה שהערך של החוליה first יהיה מטיפוס int נרשום: **Node<Integer> first = new Node<Integer>(num);** .

  ההמרות מטיפוסים לעצמים עוטפים, וחזרה, נעשות אוטומטית על ידי התוכנה.
- הערך של החוליה יכול גם להיות הפניה לעצם שהוגדר על ידנו. למשל, הביטוי: **Node <Team> t;** מגדיר את t כמשתנה שתהיה בו הפניה לחוליה גנרית שבתכונה value שלה תהיה הפניה לעצם מטיפוס Team. והביטוי **t = new Node <Team> (new Team (name));** דואג לבניית חוליה שבתכונה value שלה יש הפניה לעצם חדש מטיפוס Team, ומציב הפניה לחוליה החדשה במשתנה t.

## שמירה דינמית של נתונים

- כאשר רוצים לשמור מספר לא ידוע של נתונים (למשל, אינפורמציה עבור תלמידים כאשר לא ידוע מספר התלמידים), יש צורך להשתמש **במבנה נתונים דימני**.

  אפשרות אחת: **מערך דימני** שניתן לשנות את גודלו בזמן הרצת התכנית.
- בג'אווה אין אפשרות להגדיר מערך דינמי. כדי להגדיל מערך, אפשר לבנות מערך חדש גדול יותר מהמערך הישן, להעתיק לתחילתו את המערך הישן, ולשנות את המשתנה בו מוחזקת ההפניה למערך הישן כך שתוחזק בו ההפניה למערך החדש. למשל,

```
Shop[ ] temp = new Shop [allShops.length + 10];
for (int i = 0; i < allShops.length; i++)
        temp[i] = allShops [i];
allShops = temp;
```

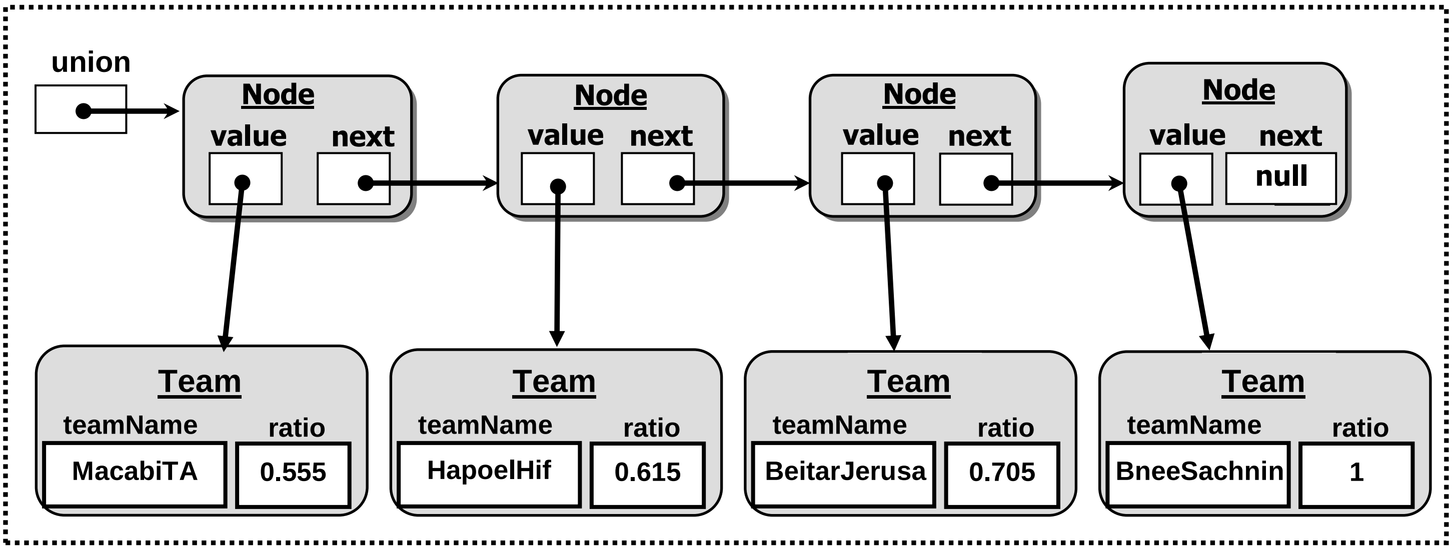
temp - מערך חדש בעל 10 איברים יותר מאשר ב allShops . בכל איבר תהיה הפניה לעצם מטיפוס Shop.

לולאה להעתקת הערכים שבמערך allShop לאיברים הראשונים של המערך temp.

המשתנה allShops מקבל את ההפניה למערך temp


© כל הזכויות שמורות, תמר פז

אפשרות שניה: **שרשרת חוליות**: אוסף של חוליות שבתכונה next של כל אחת מהן יש הפניה לחוליה הבאה. ובתכונה value נשמרת הפניה לעצם המתאים (למשל Team).




**סריקה של שרשרת חוליות:**

```
while (pos!=null)
{
    System.out.println (pos.getValue().toString());
    pos=pos.getNext();
}
```

pos הוא משתנה שבזמן הכניסה ללולאה מכיל הפניה לחוליה ראשונה בשרשרת.

כל זמן שהערך שב- pos שונה מ- null, מטפלים בחוליה הנוכחית (במקרה שלנו, מדפיסים את ערכה), ולאחר מכן מקדמים את pos כך שיכיל את ההפניה לחוליה הבאה בשרשרת (מציבים בו את ההפניה שנמצאת בתכונה next של החוליה הנוכחית).


© כל הזכויות שמורות, תמר פז

# ירושה

המחלקה Plant היא מחלקה רגילה שמגדירה עצם מטיפוס "שתיל". המחלקה Tree היא **תת מחלקה** של המחלקה Plant.

לעצם מטיפוס המחלקה Tree יש את כל התכונות שיש לעצם מטיפוס המחלקה Plant בתוספת התכונות שמוגדרות במחלקה Tree.
ובאותו אופן, על עצם מטיפוס המחלקה Tree ניתן להפעיל את כל הפעולות שמוגדרות בשתי המחלקות: Tree ו- Plant.
המחלקה Plant נקראת **מחלקה מורישה** או **מחלקת על**, והמחלקה Tree נקראת **מחלקה יורשת** או **תת מחלקה**.

```
public class Plant
{
    private String name;
    private double height;

    public Plant(String name, double height)
    {
       this.name=name;
       this.height=height;
     }

   public void addHeight (double num)
    {
       this.height=this.height+num;
    }

    public String toString()
    {
       return "name="+this.name+ " height="+this.height;
    }
}
```

```
public class Tree extends Plant
{
   private boolean eatable;

   public Tree(String name, double height ,boolean eat)
   {
         super (name, height);
         this.eatable=eat;
   }

   public String toString()
   {
      return super.toString()+ " eatable="+this.eatable;
    }
}
```

- המילה **extend** בהגדרת המחלקה Tree מציינת כי Tree היא תת מחלקה של המחלקה Plant.
- המילה **super** מאפשרת להפעיל בתוך המחלקה היורשת (Tree) פעולות (כולל הפעולה הבונה) שהוגדרו במחלקה המורישה (Plant). לכן, כאשר בפעולה הבונה של המחלקה היורשת מופיעה ההוראה: **super (*פרמטרים*)**
המשמעות היא הפעלת הפעולה הבונה של מחלקת העל (המחלקה המורישה). וכאשר במחלקה היורשת מופיעה ההוראה: **super.*פעולה* (*פרמטרים*)** (למשל (super.toString() המשמעות היא הפעלת ה*פעולה* במחלקת העל.
- הפעלת הפעולה הבונה של המחלקה המורישה, חייבת להיות ההוראה הראשונה בפעולה הבונה של המחלקה היורשת. זיכרו: "עץ הוא קודם כל צמח".
- כדי להפעיל במחלקת העל פעולה (שאיננה הפעולה הבונה) יש צורך להשתמש במילה super באחד מהמקרים הבאים:
    - כאשר הפעולה מוגדרת כבעלת הרשאה פרטית private    ( private String toString())
    - כאשר הפעולה מוגדרת עם אותה חתימה גם במחלקת העל וגם במחלקה היורשת.


© כל הזכויות שמורות, תמר פז

# הרשאת גישה protected

כאשר תכונה או פעולה היא בעלת הרשאת גישה פרטית private, אפשר לגשת אליה רק מהמחלקה בה היא מוגדרת. לכן, אם נגדיר את התכונות של המחלקה המורישה כבעלות הרשאת גישה פרטית, לא נוכל להשתמש בהן במחלקה היורשת. למשל, במקרה שלנו, לא נוכל לכתוב במחלקה Tree פעולה שתשתמש בתכונה height. מאידך, כאשר תכונה או פעולה היא בעלת הרשאת גישה פומבית public, אפשר לגשת אליה גם ממחלקות אחרות. לכן, נוכל להגדיר את התכונות כבעלות הרשאת גישה פומבית public אבל אז כל מי שישתמש במחלקה Plant יוכל לשנות את הערך שלהן!

הרשאת גישה protected (שמור) היא הרשאת גישה שמאפשרת רק למחלקה היורשת "לגעת" בתכונות של המחלקה המורישה. ובאופן כללי, לאיבר בעל הרשאת גישה protected ניתן לגשת:

- מתוך המחלקה בה הוא מוגדר (כמו בהרשאת גישה פרטית).
- מכל תת מחלקה (מחלקה יורשת) של המחלקה בה הוא מוגדר וממחלקה יורשת שלה וכן הלאה..

**הגדרת תכונות של מחלקת העל כבעלות הרשאת גישה protected :**

```
public class Plant
{
   protected String name;
   protected double height;
   ………………..
```

**הגדרת פעולה במחלקת העל כבעלת הרשאת גישה protected :**

במחלקה plant :

```
protected void addHeight (double num)
 {
     this.height=this.height+num;
 }
```

שימוש במחלקה Tree (המחלקה היורשת) בפעולה שהוגדרת כבעלת הרשאת גישה protected:

```
public void grow (double num)
 {
      if (num>1) this.addHeight (num);
 }
```

© כל הזכויות שמורות, תמר פז 

# הדפסה מעוצבת

המטרה להדפיס בצורה יפה כמו טבלה. למשל כך:

| First name | Family name | Phone | ID |
|---|---|---|---|
| Gil | Paz | 0542227328 | 1234567890 |
| Dan | Daniel | 0501234663 | 3033030399 |
| Yael | Lev | 0303454575 | 3044545474 |

**ההוראה String.format ("%-13s",item) יוצרת מחרוזת שארכה 13 תווים. המחרוזת תכיל את item ולאחריו רווחים.**

הערות:

1. item יכול להיות שם של משתנה מכל טיפוס שהוא (כולל מספר לא שלם). ויכול גם להיות ערך כלשהו מכל טיפוס.
2. אפשר כמובן לבחור אורך אחר (לא 13 ☺ )
3. בלי – (למשל "%13s") המחרוזת תכיל רווחים ולאחריהם את item.

**הוראת ההדפסה** System.out.println(st) כאשר st הוא מטיפוס String.format תיצור הדפסה מעוצבת כמו טבלה. למשל הקטע הבא:

```
String fn = String.format ("%-13s","first name");
String sn = String.format ("%-13s","second name");
System.out.println (fn+sn);
```

ייצור את ההדפסה:

```
first name   second name
```

בהדפסה האחרונה בטוח כי המילה second תתחיל בדיוק 13 תווים לאחר תחילת המילה first (כי דאגנו שהאורך של first name יהיה 13 תווים בדיוק).

## למה זה טוב?

נניח שיש לנו עצם Student שהתכונות שלו הן שם, שם משפחה, מספר טלפון ומספר תעודת זהות. במחלקה אחרת יש לנו שרשרת של עצמים מטיפוס Student ונרצה להדפיס את הפרטים של כל ה Student-ים. כדי ליצור הדפסה יפה, נשנה את פעולת toString של Student כך שתהפוך את כל התכונות ל String.format ותיצור מהן מחרוזת אחת. למשל כך:


© כל הזכויות שמורות, תמר פז

```
public String toString()
{
  return String.format ("%-13s", this.firstName)+
          String.format ("%-13s", this.familyName)+
          String.format ("%-13s", this.phoneNum) +
          String.format ("%-13s", this.idNum);
}
```

במחלקה האחרת (שבה שרשרת ה Student) נכתוב את הפעולה הבאה שתדאג להדפסה המעוצבת:

```
{
   String firstn = String.format ("%-13s","first name");
   String familyn = String.format ("%-13s","family name");
   String phone = String.format ("%-13s","phone number");
   String id = String.format ("%-13s","ID number");
   System.out.println (firstn+ family+ phone+id);
   //    עד פה דאגנו להדפסת הכותרת ועכשיו נעבור על השרשרת ונדפיס את התכונות של העצמים
   pos = this.allStudents;
   while (pos!=null)
      System.out.println (pos.getValue().toString());
}
```

כמובן שאפשר שהמחרוזות תהיינה באורכים שונים. למשל firstName יהיה באורך 13 ו secondName באורך 20


© כל הזכויות שמורות, תמר פז