מבוא לחישוב שיעור מספר 9

תכנות מונחה עצמים 1:

בשיעור זה נתחיל נושא חדש: תכנות מונחה עצמים, נלמד לבנות טיפוסים (מחלקות) חדשים, ליצור מהם אובייקטים, נדגים זאת ע"י כתיבה של שתי מחלקות נקודה, ואוסף של נקודות. נכיר את החלקים השונים במחלקה: מידע, בנאים, שיטות. נעמיק באופן שבו מיוצגים האובייקטים השונים, מנגנון הקריאה לשיטות (בשונה מקריאה לפונקציות סטטיות).
בחלק השני של השיעור נתחיל להבין את מנגנון הירושה ב java (מקרה פרטי של הכלה), ונדגים אותו ע"י מספר דוגמאות.

מבוא כללי:
עד עתה השתמשנו בשני מינים של טיפוסים: פרימיטיביים (נניח .. int, boolean) וטיפוסים מורכבים שמוגדרים ב java (נניח מחרוזות ומערכים), תכנות מונחה עצמים בעצם מאפשר למתכנת להגדיר טיפוסי נתונים בעצמו (מחלקות), מהם יוכל לצור עצמים (אובייקטים).

"מחלקה" היא הגדרת טיפוס חדש המאפשר ריכוז של נתונים ופונקציות תחת שם אחד. מבנה המחלקה מוגדר ע"י המידע (משתנים) אשר היא מכילה ואילו יכולות (פונקציות) יכולה המחלקה לבצע.

לאחר שמגדירים מחלקה ניתן להגדיר ולממש אובייקטים מהטיפוס של המחלקה כשם שניתן להגדיר משתנים מטיפוסים בסיסיים. הפונקציות שייכות למחלקה ולכן כאשר אובייקטים שונים מפעילים פונקציה, מופעלת הפונקציה של המחלקה שלהם. ולכן בכל פעם תתבצע פעולה שונה, ומכאן השם תכנות מונחה עצמים.

דוגמא: המחלקה נקודה: איזה מידע, אילו יכולות, למה כדאי??

מחלקה – Class

תבנית הגדרת מחלקה:

```
class < שם המחלקה > {
    <משתני עצם>
        <בונים>
        <שיטות>
}
```

הגדרת מחלקה יוצרת טיפוס חדש מסוג <שם המחלקה> וניתן ליצור משתנים מהטיפוס הזה.

משתני עצם
1. מוגדרים בתוך ה class
2. כל הפונקציות מכירות אותם
3. כל פונקציה יכולה לשנותם

נדגיש את ההבדל בין משתני העצם למשתנים הלוקליים (עליהם דיברנו בשיעורים האחרונים).
משתנים לוקליים:
1. מוגדרים בתוך הפונקציה
2. מוכרים רק בפונקציה בה הוגדרו
3. כדי להשתמש בהם בפונקציה אחרת יש להעבירם כפרמטרים

דוגמא: נקודה:

```
class Point{
    public double xVal;
```

```
    public double yVal;
```

עכשיו אנחנו יכולים (למשל בתוך ה main) לכתוב:

```
Point myPoint = new Point();
```

כמו שכתבנו קודם:

```
int myInt = 7;
```

יצרנו אובייקט מסוג Point בשם myPoint.

אובייקטים מאפשרים לנו:

1. לאגד כמה נתונים תחת הגדרה אחת
2. ליצור טיפוסים מורכבים בניגוד לטיפוסים פרימיטיביים שהכרנו עד עכשיו (int, char…)
3. להגדיר פעולות על טיפוסים אלה
4. להגן על המידע

נבדיל כאן בין מחלקה ואובייקט. מחלקה היא ההגדרה שאנו כותבים. זוהי הדרך להגדיר טיפוס חדש. אובייקט הוא עצם שנוצר מטיפוס המחלקה. נהוג לדבר על המחלקה כעל תבנית ועל כל אובייקט כעל תוצר של התבנית.

קיבלנו חבילה שמכילה 2 מספרים ומייצגת נקודה. אנחנו יכולים לגשת למשתנים שלה, למשל:

```
myPoint.xVal = 10;
myPoint.yVal = myPoint.xVal;
```

שיטות

מחלקה יכולה יותר מאשר רק להכיל מידע. היא גם מאפשרת לבצע כל מיני חישובים על המידע הזה.
נוסיף 2 פונקציות למחלקה:

1. הדפסת נתוני הנקודה
2. שינוי ערכי הנקודה

```
class Point{

    public double xVal;
    public double yVal;

    /******* Methods *****/

    public void printPoint() {
        System.out.println ("Point: x="+xVal+" y="+yVal);
    }

    public void setValues(int xVal, int yVal){
          this.xVal = xVal;
          this.yVal = yVal;
    }
} // class Point
```

עכשיו נוכל לכתוב :

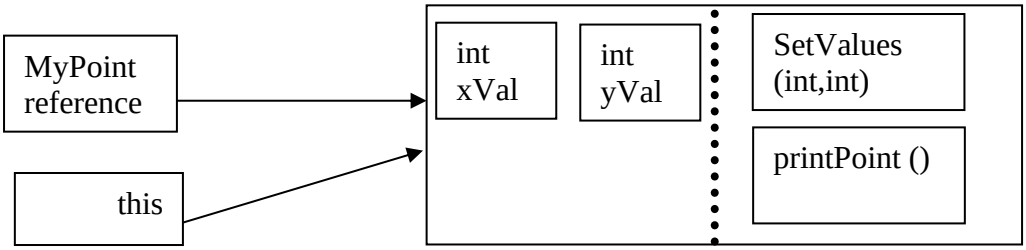
```
Point myPoint = new Point();
myPoint.SetValues(5, 10);
myPoint.printPoint();
```

הערות :

1. המשתנים xVal yVal הם משתנים גלובליים של המחלקה.
2. כאשר אובייקט משתמש במלה השמורה **this** הוא מקבל reference לעצמו.
כלומר, המלה השמורה **this** מתייחסת לאובייקט הספציפי שעליו אנחנו עובדים.

- דיון על this.

התמונה בזיכרון:



פונקציה חשובה לכל אובייקט היא toString . נחזור למחלקה Point:

```
class Point{

    public String toString () {
        return "point: x = " + xVal + "y = " + yVal;
    }

}
```

הפונקציה toString() מחזירה String המייצג את האובייקט. כאשר קוראים לפונקציה System.out.print() עם אובייקט, נקראת השיטה toString( ) באופן אוטומטי. אם לא ממשנו את השיטה יודפס על המסך כתובת האובייקט בזיכרון.

<u>השיטה הבונה - constructor</u>

המלה השמורה new גרמה להקצאת מקום בזיכרון עם 2 integers ו- 2 שיטות.
המלה השמורה new גם קוראת לשיטה מסוימת במחלקה: הבנאי. זוהי שיטה מיוחדת שמגדירה את אתחול האובייקט. הפקודות הרשומות בתוך הבנאי יופעלו אוטומטית עם יצירת האובייקט.

```
class Point{
    public double xVal;
    public double yVal;

    Point (double xVal, doubleyVal){
        this.xVal = xVal;
        this.yVal = yVal;
    }

    Point(){
        this.xVal =0;
        this.yVal = 0;
    }
}
```

- לשיטה בנאית יש אותו שם כמו למחלקה שלה, והיא לא מחזירה ערך.
- בנינו כאן 2 בנאים שמקבלים פרמטרים שונים. את זה אפשר לעשות עם כל פונקציה וזה נקרא **method overloading**.

המשמעות היא שניתן לבנות נקודה בשתי דרכים:

1. עם ערכי x y.
2. בלעדיהם ואז נקבל את נקודת ברירת המחדל.

כזכור: לא ניתן להגדיר 2 פונקציות שההבדל היחיד ביניהן הוא בערך ההחזרה.

כדי ליצור נקודה בעזרת הבונה השני נרשום:

```
Point myPoint = new Point(4.7, 9);
```

- כשמגדירים מחלקה נוצר לה אוטומטית constructor ללא פרמטרים שלא מבצע כלום. אם מגדירים למחלקה constructor אחר אז deafult constructor נמחק.

תכונה נוספת של אובייקטים ב java היא שהשיטות שלהם יכולות לקבל .
למשל נרשום בתוך המחלקה Point את השיטה הבאה:

```
public double distance(Point otherPoint){
        double yDiff = yVal-otherPoint.yVal;
        double xDiff = xVal-otherPoint.xVal;
        return Math.sqrt(ydiff*yDiff+ xDiff*xDiff);
}
```

- ראינו 2 בנאים של המחלקה Point נבדוק כעת בנאי שלישי:

```
class Point{

    // Everything we did last time

    Point(Point p){
        xVal = p.xVal;
        yVal = p.yVal;
    }
}
```

בנאי זה נקרא: copy constructor
דרך אחרת לכתוב את אותו בנאי:

```
    Point(Point p){
        this(p.xVal,p.yVal);
    }
```

String הוא דוגמא לעצם קיים. הוא מכיל שיטות, ובונים.
ניתן לגשת ל – API (Application Programming Interface) בכדי לראות את תכולת המחלקות השונות.

Modifiers - הרשאות : private ו – public.

נסביר עתה את משמעות המילים השמורות : private, public.
משתנה או פונקציה שהוגדרו להיות private הם פרטיים ויוכלו להשתמש בהם רק הפונקציות של המחלקה.
משתנה או פונקציה שהוגדרו להיות public הם ציבוריים ויוכרו ע"י כלל הפונקציות.
לנתונים במחלקה אפשר לתת רמות שונות של אפשרות גישה ממחלקות אחרות.

למשל נכתוב מחלקה Car:

```
class Car{
	private String color;
	private double speed;
	private int year;

	public color getColor(){
		return color;
	}

	public void setColor(String color){
		if (!color.equals("gold")){
			this.color = color;
	}

	public double getSpeed(){
		return speed;
	}

	public void setSpeed(double speed){
		if (speed <= Law.SPEED_LINIT)
			this.speed = speed;
	}
}
```

סיבות לשימוש בהרשאות:

1. יצירת ריכוזיות (קופסא שחורה)
2. הגנה על משתנים
3. מאפשר בדיקה של הפעולות.

דוגמא נוספת היא אם אנחנו רוצים לאפשר מניפולציות על משתנים אבל לא לאפשר לראות אותם. למשל – דוגמת הכספת:

```
class Safe {
	public int doorNumber = 123;
	private boolean locked = true;
	private int combination = 456;

	public boolean isLocked () {
		return (locked);
	}

	public void unLock (int thisCombination) {
		if (thisCombination == combination) unLock();
	}

	private void unLock() {
		locked = false;
	}

	private void setCombination (int setting) {
		combination = settings;
	}


	Safe (int door) {
		doorNumber = door;
		SetCombination (doorNumber);
	}
```

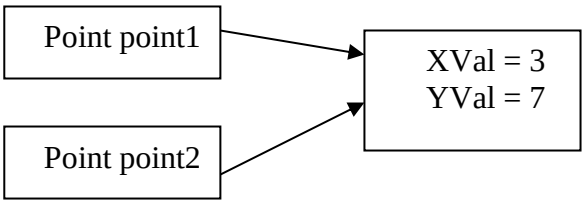
```
        Safe (int door, int combination)  {
                doorNumber = door;
                this.combination  =  combination;
        }


}
```

המשתנים וגם חלק מהפונקציות הם פרטיים ולכן רק מתוך המחלקה ניתן לגשת אליהם.
נשים לב ל overloading של הבנאים.

<u>אוביקטים וזיכרון</u>

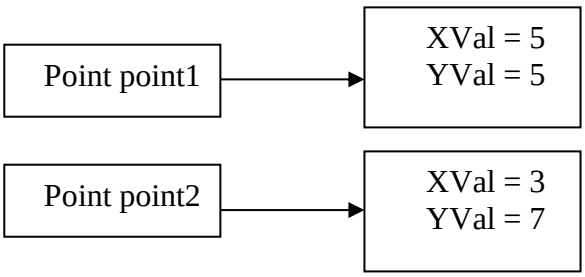
כמו מערכים (שהם בעצם סוג של אוביקט) גם אוביקטים מועברים by reference. המשמעות:

```
Point point1 = newPoint();
Point point2 = point1;
point1.SetValues(3.0, 7.0);
print(point2);

//output:
// Point: x=3.0 y=7.0
```



לעומת זאת:

```
Point point1 = new Point(3.0, 7.0);
Point point2 = new Point(point1);
point1.SetValues(5.0, 5.0);
print(point1);
print(point2);

//output:
// Point: x=5.0  y=5.0
// Point: x=3.0  y=7.0
```



פונקציות שמקבלות אוביקטים ומשנות אותם – השינוי נשמר גם מחוץ לפונקציה (כמו במערכים). נגדיר מחלקה:

```
class Number{
        int num;

        Number(int num){
                this.num=num;
        }

        public void setNum(int newNum){
                this.num=newNum;
        }

        public String toString(){
                return “number = “ + num;
        }
}
```

עכשיו נניח שב- main יש לנו קריאה לפונקציה changeValues

```
public static void changeValues(Number num1, Number num2, Number num3, int num4){
        num1.num = num1.num+1;
        num2.setNum(num2.num+1);
```

```
        num3 = new Number(num3.num + 1);
        num4 = num4+1;
}

…main(){
        Number num1 = new Number(1);
        Number num2 = new Number(2);
        Number num3 = new Number(3);
        int num4 = 4;
        changeValues(num1, num2, num3, num4);
        System.out.println(num1);
        System.out.println(num2);
        System.out.println(num3);
        System.out.println(num4);
}

//results:
// 2
// 3
// 3
// 4
```

<u>סיכום מבוא לתכנות מונחי עצמים:</u>

- דרך להגדיר טיפוסים חדשים, וליצור מהם אובייקטים.
- אופן כתיבה של תוכנית: כל טיפוס (מחלקה) הינו קובץ נפרד, מאפשר גמישות בפיתוח.
- לתכנות מונחה עצמים 3 תכונות עיקרית :
  - הכמסה – **encapsulation**: מאפשרת ההפרדה בין המה לבין האיך.
  - ירושה – **inheritance**: מאפשרת גזירת מחלקות ממחלקות אחרות וע"י כך המחלקות הנגזרות יורשות את התכונות של המחלקות שמהן הן נגזרו.
  - רב צורתיות – **polymorphism**: מאפשרת כתיבת אלגוריתמים ומבני נתונים לשימוש כללי.