

סיכום שיעור

חזרנו על העקרונות של תכנות מונחה עצמים, שהוא למעשה עבודה עם מחלקות (classes) שמכילות מאפיינים (משתנים) וגם פעולות (מתודות).

כפי שכבר הזכרנו בעבר, מחלקות נועדו בעיקר להגדיר טיפוסים נתונים מורכבים שאינם נימצאים בשפה (C#), כלומר מחלקה היא מעין 'פס ייצור', אם תרצו, שממנו ניתן להגדיר עצמים, ולכל עצם שמוגדר בעזרת המחלקה, יהיה סט של מאפיינים ואפשרות להפעלת כל המתודות עליו, בשעת הצורך.

הגדרנו בשיעור מחלקה מסוג תיבה שבה המאפיינים היו אורך, רוחב וגובה של מקצועות התיבה. הפעולות שאותן הגדרנו היו לאחזר את הנפח, את השטח, והדפסה של מימדי התיבה.

הבונה היה פשוט, השמה של הערכים על פי סדר של קבלת הפרמטרים עבור המקצועות (אורך, רוחב וגובה).

הנה המחלקה שיצרנו:

```
class Box {

    private double length;
    private double width;
    private double height;

    //constructor

    public Box(double a=1, double b=1, double c=1){
        length=a;
        width=b;
        height=c;
    }

    public void print(){
        Console.WriteLine("(" + length + ";" + width + ";" +
            height+"");
    }

    public double volume(){
        return length * width * height;
    }

    public double surfacearea(){
        return (length * height * 2 + width * height * 2 +
            length * width * 2);
    }
}
```

וכעת ניראה שימוש לדוגמה במחלקה הנ"ל (נגדיר עצם בודד וגם מערך של עצמים):

```
class Mainclass{
    public static void Main(string [ ] args){
        Box b1 = new Box(12,10,8); //Create an object b1 which is a box
        Console.WriteLine(b1.volume()); // will print (12; 10; 8)
        //Now defining an array of objects of type: Box
        Box []list= new Box[10]; //The 'new' here is the array of objects
        for(int i=0;i<10;++i){
            list[i]=new Box(i,i,i); //The 'new' here is each individual Box
            list[i].print();
            Console.WriteLine("surface area is: "+ list[i].surfacearea());
        }
    }
}
```

בנוסף, דיברנו על שלושת הרעיונות המרכזיים מאחורי תכנות מונחה עצמים (oop): הסתרת מידע (Data encapsulation), הורשה (inheritance) וריבוי צורות (polymorphism) הסתרת עד היום בעיקר ראינו את הדבר הראשון: הסתרת מידע בתוך מחלקה ושימוש בו בתוכנית ה-Main.

דוגמה להורשה וריבוי צורות. נגדיר מחלקה שניקראת: Cube. זוהי מחלקה שיכולה ליצור עצמים מסוג קוביה שזה מקרה פרטי של תיבה (תיבה שבה המקצועות שווים כולם). היא תירש את מה שיש ב- Box ותוסיף עוד דברים משלה.

```
class Cube:Box{
    private string name = "Cube"; //added a property
    //A derived class does not inherit the constructor
    //Since Cube is derived from Box, and Box does not
    //have a default constructor, Cube must invoke base.

    public Cube(double x, double y, double z):base(x,y,z){
        if (x != y || x != z)
            Console.WriteLine("error");
    }
}
```

על מנת להגדיר מתודה בשם: print שתדפיס במקרה של קוביה גם את המילה: "קוביה", עלינו לשנות 2 דברים במה שעשינו עד כה. 1) להגדיר את המאפיינים כ-protected (במקום: private) ולהגדיר את print במחלקה Box כ- virtual

```
public override void print(){
    Console.WriteLine("(" + length + ";" + width + ";" +
        height+")" + " "+name);
}
}
```

הסבר על: Cube

1. התחביר עבור הורשה זה לאחר המילים: class ושם המחלקה החדשה, כותבים נקודותיים ואת שם המחלקה ממנה יורשים. בדוגמה הנ"ל זה: `class Cube:Box`

2. כתוצאה מההורשה, גם במחלקה החדשה יש את שלושת המאפיינים: length, width, height וגם את המתודות: volume, surfacearea, print. (זאת מבלי שהגדרנו אותם ב-Cube). בנוסף, הגדרנו את המאפיין name שאינו קיים ב-Box.

3. הבונה (constructor) של Cube הוא מחוייב המציאות, כיוון ש-Cube אינה יורשת את הבונה של Box. שימו לב שאחרי הפרמטרים של הבונה של Cube, כתבנו `:base(x,y,z)`

זהו התחביר שמאפשר הפעלת הבונה של Box על הפרמטרים של Cube שהם: x, y, z

כלומר לפני כתיבת הפרטים של הבונה של Cube, אנו מפעילים את הבונה של Box, כי גם ב-Cube אנו רוצים לאתחל את שלושת המקצועות באותה צורה. כדי להשאיר הכל פשוט, בבונה של Cube יש לנו רק בדיקה שהמקצועות שונים והדפסת שגיאה אם אינם שונים

4. כדי להדגים את ריבוי הצורות (polymorphism) היינו צריכים לשנות את ההגדרות של המאפיינים של Box בצורה הבאה:

```
protected double length;
protected double width;
protected double height;
```

בנוסף, נשנה את ההגדרה של: `print` בתוך Box להיות וירטואלית בצורה הבאה:

```
public virtual void print(){
    Console.WriteLine("(" + length + ";" + width + ";" +
        height+");");
}
```

5. זה שכת `print` היא וירטואלית מאפשר לנו ל"העמיס" עליה את הגירסה השונה במקצת שאותה הגדרנו בתוך: Cube (בעזרת הפקודה: `override`). מנגנון זה מאפשר לנו, להפעיל את `print` על עצמים מסוג Box וגם עצמים מסוג Cube, והיא תתנהג בהתאם לסוג העצם עליו היא פועלת בזמן הריצה של התוכנית. הנה דוגמה להפעלת ההורשה וריבוי הצורות.

```
Cube a = new Cube(3, 3, 3); // הכל בסדר - יצירת קוביה
Cube b = new Cube(3, 3, 7); // error
```

בפקודה השניה הוא ידפיס error כי הבונה של Cube 'יגלה' שהצלעות אינן שוות. מאוחר יותר ניראה כיצד למנוע את יצירת הקוביה במיקרה כזה, בינתיים רק נדפיס הודעה.

כעת נגדיר מערך מסוג Box ונשים בו תיבה אחת וקובייה אחת (הגדרה של אובייקט יותר כללי יכול לקבל אחד יותר פרטני)

```
Box []boxcube= new Box[2];
boxcube[0] = new Box(1,2,3);
boxcube[1] = a; //we defined it above

for (int j = 0; j < 2; ++j) {
    boxcube[j].print();
}
```

לולאת ה-for תדפיס גם את התיבה וגם את הקובייה, בעזרת הגירסה המתאימה לכל אחת של: print, והתוצאה תראה כך:

```
(1;2;3)
(3;3;3) Cube
```

איך יודעים זאת? המילה Cube מודפסת רק במתודת ה-print שנימצאת במחלקה: Cube

מחלקת שירות

זו מחלקה שמאגדת מתודות סטטיות שבהן ניתן להשתמש ללא הגדרת עצמים, בדומה למחלקה Math שבה כבר השתמשנו. למשל כדי למצוא שורש ריבועי של a אפשר לכתוב:

```
b=Math.sqrt(a);
```

שיעורי הבית היו לכתוב מחלקת שירות שבה מתודות שמטפלות במערכים. (עמוד 154 תרגיל 1 בספר)