

מחלקות

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

```
public class המחלקה
```

```
{
    // הגדרת התכונות של המחלקה
    private שם תכונה1 טיפוס תכונה1 ;
    private שם תכונה2 טיפוס תכונה2 ;
    .....
    private שם תכונהח טיפוס תכונהח ;
    // הגדרת הפעולה הבונה -- הפעולה הבונה מחזירה עצם מטיפוס המחלקה
    public המחלקה (צרך טיפוס1, צרך טיפוס2, .....)
```

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

= public ניתן להשתמש במחלקה זו ממחלקות אחרות

```
{
    this.תכונה1 = ערך;
    this.תכונה2 = ערך;
    .....
    this.תכונהח = ערך;
}
```

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 המחלקה (צרך טיפוס1, צרך טיפוס2, ..... טיפוס הצרך המחזיר)
{
    הוראות לביצוע
}
```

public = הרשאת 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 המחלקה ()`
- חתימה של פעולה (שונה מהפעולה הבונה) שלא מקבלת ערכים:

`public המחלקה (טיפוס הצרך המחזיר הרשאת גישה)`

דוגמא: `public int GetDay ()`

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

- 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, ההתייחסות לתכונה של העצם עליו מתבצעת הפעולה, היא באמצעות המילה **this**. המילה **this** מציינת את העצם הנוכחי עליו מתבצעת הפעולה.
- C# 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:

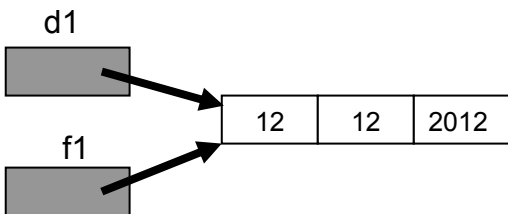

```
public Date (int day , int month , int year)
```

 החתימה של הפעולה השנייה תהיה:


```
public Date( )
```

 המנגנון שמאפשר להגדיר מספר פעולות בעלות אותו שם נקרא **העמסה (overloading)**.

```
namespace DateProject
{
class Program
{
public static void main(String[] args)
{
Date d1 = new Date (10,9,2006);
Console.WriteLine ("before test d1 = " + d1.ToString());
Test (d1);
Console.WriteLine ("after test d1 = " + d1.ToString());
}
private static void Test (Date f1)
{
f1.SetDay (12);
f1.SetMonth (12);
f1.SetYear (2012);
Console.WriteLine ("in test f1 = " + f1.ToString());
}
}
}
```



The diagram illustrates the state of the Date object after the Test method is called. It shows two variables, d1 and f1, each represented by a grey rectangular box. Arrows from both boxes point to a single Date object, which is represented as a horizontal table with three columns containing the values 12, 12, and 2012. This indicates that both variables now reference the same Date object with the updated values.

- **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