מחלקות ועצמים

מהי מחלקה (Class) ב-C#?

מחלקה היא תבנית או שבלונה ליצירת עצמים (Objects).
היא מתארת:

- תכונות (Fields / Properties) — מה האובייקט "יודע" לשמור.
- פעולות (Methods) — מה האובייקט "יודע" לעשות.

לדוגמה:
מחלקה בשם Car יכולה לתאר את המאפיינים של רכב:

- תכונות: צבע, מהירות מרבית, יצרן.
- פעולות: נסיעה, עצירה.

מהו עצם (Object) ב-C#?

עצם הוא מופע ממשי של מחלקה.

ביצירת מחלקה נוצר כביכול "טיפוס" חדש, בדומה לטיפוסים שאנחנו כבר מכירות: int, double וכו', רק שכאן מדובר בטיפוס מורכב יותר, שיכול להכיל כמה ערכים מטיפוסים שונים-שהם בעצם התכונות.

ביצירת עצם-אובייקט נוצר "משתנה" מהטיפוס שהגדרנו.

דימוי פשוט:

- מחלקה היא כמו תוכנית בנייה של בית.
- עצם הוא בית אמיתי שבנו לפי התוכנית.

תכלס בקוד:

הגדרת מחלקה, בעזרת המילה השמורה class, והרשאת הגישה public ציבורית:

public class שם המחלקה

ובתוך המחלקה, נגדיר את התכונות-מאפיינים, ופעולות של המחלקה.

הגדרת תכונה:

private טיפוס שם התכונה;

*שם מחלקה יתחיל באות גדולה.

פעולה בונה (Constructor)

פעולה בונה היא פעולה מיוחדת שנקראת אוטומטית כאשר יוצרים מופע חדש של מחלקה (new ClassName()).

מטרתה:

- לאתחל ערכים של שדות
- להכין את האובייקט לשימוש תקין

תכונות עיקריות של פעולה בונה:

- השם שלה חייב להיות כמו שם המחלקה בדיוק.
- אין לה טיפוס החזרה (לא void ולא שום דבר אחר).
- אפשר להגדיר כמה פעולות בונות (כל אחת עם חתימה שונה = Overloading).

דוגמא:

```
public class Game
{
    public string title;
    public int maxPlayers;

    // פעולה בונה ברירת מחדל
    public Game()
    {
        this.title = "Unknown Game";
        this.maxPlayers = 2;
    }

    // פעולה בונה עם פרמטרים
    public Game(string title, int maxPlayers)
    {
        this.title = title;
        this.maxPlayers = maxPlayers;
    }
}
```

דוגמאות שימוש בקוד:

1. יצירת עצם מטיפוס Game עם הפעולה הבונה ברירת מחדל:

```
Game defaultGame = new Game();
```

הסבר: מאחר ולא שלחנו פרמטרים, האובייקט יקבל ערכים שהוגדרו בפעולה הבונה ברירת המחדל.

2. יצירת משחק עם ערכים מותאמים:

```
Game customGame = new Game("Soccer", 22);
```

הסבר: כאן שלחנו ערכים ספציפיים והפעולה הבונה עם פרמטרים קיבלה אותם והכניסה לשדות.

this

מילת מפתח ב C# שמייצגת את **האובייקט הנוכחי** שבו אנחנו נמצאים.
כלומר — אם את נמצאת בתוך מחלקה ומבצעת פעולה (למשל פעולה בונה או פונקציה), this מתייחסת לאותו מופע (אובייקט) שנוצר ממחלקה זו.

למה משתמשים ב-this?

בעיקר כדי להבדיל בין:

- שם של **פרמטר** בפעולה (למשל קלט שקיבלת)
- לבין **שדה** (field) של המחלקה.

לדוגמא:

```
public class Book
{
    private string title;

    public Book(string title)
    {
        this.title = title;  // השדה הפרטי במחלקה = הפרמטר שהתקבל
    }
}
```

בלי this, המחשב לא היה יודע אם הכוונה לפרמטר או לשדה — כי לשניהם קוראים באותו שם.

פעולות Get ו-Set

כאשר יש תכונות private במחלקה (כלומר — לא נגישות מבחוץ), אנחנו יוצרים פעולות גישה (Getters) ופעולות שינוי (Setters) כדי לאפשר עבודה מבוקרת עם השדות האלה.

לדוגמא:

```
private int pages; // שדה פרטי

public int GetPages()
{
    return this.pages; // פעולה שמחזירה את מספר העמודים
}

public void SetPages(int pages)
{
    this.pages = pages; // פעולה שמעדכנת את מספר העמודים
}
```

מתי משתמשים ב-Get ו-Set?

- אם רוצים **לקבל** ערך — קוראים ל-()GetPages.
- אם רוצים **לשנות** ערך — קוראים ל-(5)SetPages, לדוגמה.

למה לא לאפשר גישה ישירה?

- כדי לשמור על **בקרת גישה**: למשל, בעתיד אפשר להוסיף תנאים בפעולות (לדוגמה: לא לאפשר מספר עמודים שלילי).
- כדי לקיים את עקרון **הכימוס** (Encapsulation) – מחביאים את פרטי המימוש מהמשתמש החיצוני.

ToString()

המטרה של הפעולה היא להחזיר מחרוזת המייצגת את האובייקט.

הפעולה מחזירה מחרוזת המורכבת מצמדים של שם תכונה+ערך תכונה לכל אחת מהתכונות. לדו':

```
public string ToString()
{
    return $"Title: {title}, Author: {author}, Pages: {pages}";
}
```

הבדלים בין פעולות סטטיות לפעולות פנימיות (לא סטטיות)

פעולות פנימיות (לא סטטיות):

- פעולה רגילה (כמו GetPages, SetPages).
- קשורה לאובייקט ספציפי — חייבים ליצור מופע (new) של המחלקה כדי להשתמש בה.
- יש לה גישה לתכונות של האובייקט.

לדוגמא:



הפעולה GetPages היא פעולה פנימית, וניתן לראות כי היא מופעלת על אובייקט-עצם.

פעולות סטטיות (static):

- פעולה השייכת למחלקה עצמה ולא לאובייקט מסוים.
- לא צריכה אובייקט כדי לקרוא לה.
- לא יכולה לגשת לתכונות רגילות (כי אין לה אובייקט ספציפי).