

תכנות מונחה עצמים

הגדרת סוגים או טיפוסים שונים מאלה הקיימים בשפה (int, bool, string וכו') למה זה טוב? מיועד לנסות ולפשט את התהליך של בניית תוכניות מורכבות. האם זה שונה ממה שעשינו עד כה? לא ממש, זה רק עוד נידבך מסייע לתכנות.

הזכרנו את מילת המפתח **class** (מחלקה או טיפוס) שבעזרתו ניתן להגדיר עצמים (objects). הייתה לנו דוגמה של מחלקה מסוג: תלמיד.

אמרנו שעצם הוא מקרה פרטי של מחלקה, כלומר ממחלקה בשם תלמיד, ניתן לבנות עצמים, כאשר כל עצם הוא תלמיד ספציפי, ותהליך זה ניקרא: בניית עצם או: instantiation of an object

אמרנו שעצם מורכב לרוב **מתכונות** (attributes או properties) ו**מפעולות** (methods) או מתודות (methods) כאשר התכונות הן משתנים והפעולות הן פקודות ב- C# (כמו משפטי תנאי לולאות וכדומה)

דיברנו על מחלקה שניקראת: **Circle** שיכולה לשמש אותנו בתכנות גראפי לעבוד עם מעגלים.

אלה תכונות היו נחוצות עבור מעגל?

(1) נקודת מרכז

(2) הרדיוס

(3) הצבע

אלה פעולות נרצה לבצע על מעגל?

(1) לחשב שטח

(2) לחשב היקף

(3) לצייר אותו

(4) לשנות את הרדיוס (להקטין או להגדיל את המעגל)

באופן כללי אמרנו שהפעולות שנגדיר תלויות במה שנרצה לעשות עם המעגל. אם הצרכים ישתנו, נוכל להוסיף או לגרוע ממה שכבר הגדרנו. כמו שהזכרנו, התכנות הראשוני הוא 10% והשינויים ושדרוגים 90% מהעבודה (תהליך איטראטיבי).

16.1.2017

דיברנו גם על 'פעולה בונה' או בנאי (**constructor**) - פעולה מיוחדת שמגדירה את העצם החדש.

זוכרים את מילת המפתח **new** ?

אמרנו שיצירת עצם חדש נעשית תוך כדי ריצת התוכנית ולכן המחשב לא יודע מראש להקצות זיכרון עבור העצם ולכן **הקצאת הזיכרון היא 'דינמית'** (זיכרון ערימה- **Heap**)

עבור משתנים שאינם עצמים הקצאת הזיכרון נעשית לפני ריצת התוכנית (בזמן קומפילציה) - זיכרון 'מחסנית' (**Stack**)

איך יכול להיות שהמחשב יודע מראש על משתנים 'רגילים' ולא על עצמים?

(הרי הוא יודע שהגדרנו מחלקה - לא?)

סוגי גישה לתכונות ולפעולות

דיברנו על **public** ועל **private**. מה שיוגדר כ- **private** יהיה נגיש רק בתוך המחלקה ומה שמוגדר **public** בכל מקום (עוד נדבר על מה זה 'כל מקום')

אחת המטרות של תכנות מונחה עצמים זה 'להחביא' נתונים ממי של צריך לגעת בהם (**Data hiding**)

אם נקודת מרכז המעגל יכולה להשתנות רק בתוך המחלקה ופתאום זז לנו המעגל למקום לא צפוי - במקום לחפש בין 5000 שורות קוד, ניגש ישר להגדרת המחלקה (50 שורות קוד) ונברר מה עשינו לא נכון.

כללית אמרנו שהתכונות תוגדרנה כ- **private** והפעולות כ- **public**

זה מזכיר משהו?

זוכרים את המשפט ה'סתום' שמופיע בראש התוכנית ?

```

using System;
namespace ourprograms
{
    class MainClass
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("HELLO WORLD");
        }
    }
}

```

כל מה שעשינו עד כה השנה בתכנות, היה מרוכז בכתיבת 'פעולה' (הפעולה הראשית של התוכנית) כלומר יצרנו מחלקה אחת ויחידה, ובתוכה הגדרנו פעולה אחת שקראנו לה 'תוכנית'.

אמרנו שתוכנית ב- C# היא אוסף של מחלקות. עד כה הייתה לנו מחלקה אחת, כעת ניצור עוד.

פעולות - אמרנו רצף של פקודות בשפה

עבור כל פעולה או method שניכתוב בשפת C# בד"כ **בכותרת של הפעולה** ניראה מילות מפתח כמו: int, public, void וכד'. כעת הגיע הזמן להסביר מה הכוונה. למשל:

```
public static void Main(string[] args)
```

public - הגישה לפעולה מותרת מכל מחלקה בתוכנית

static - את זה נסביר בשלב מאוחר יותר

void - מילת מפתח שאומרת כי הפעולה אינה מחזירה ערך. אם היא מחזירה, כאן יהי סוג הערך.

Main - שם הפעולה. לכל פעולה יש שם ונהוג לתת שם עם אות ראשונה גדולה.

בסוגריים ישנם מה שנקרא 'פרמטרים' שהם משתנים שהפעולה צריכה כדי לעשות את עבודתה.

16.1.2017

דוגמאות נוספות להגדרת כותרות של פעולות:

public void print()

זוהי כותרת של פעולה בשם print שניתן להשתמש בה על ידי פעולות אחרות כי היא public והיא לא מחזירה שום ערך ולכן: void. היא לא מקבלת פרמטרים לכן הסוגריים ריקים.

public int show_grade(1)

זוהי כותרת של פעולה בשם: show_grade שגם ניתן להשתמש בה בכל מקום והיא מחזירה ערך שלם (int), היא גם מקבלת פרמטר (בסוגריים)

פעולה בונה - Constructor

זו פעולה שמגדירה את העצם ההתחלתי שאותו נירצה ליצור במהלך התוכנית. הפעולה הבונה תהיה **בעלת אותו שם כמו שם המחלקה** ולא יהיה לה ערך מוחזר (אפילו לא void)

נחזור למעגל

דוגמת הגדרת המחלקה של המעגל תהיינה 4 תכונות - קואורדינטת x, קואורדינטת y, רדיוס r וצבע-color

הפעולות תהיינה: ציור המעגל וחישוב שטחו (בינתיים, אחר כך נוסיף עוד)

```

public class Circle {
    private double r, x, y;    //properties
    private string color;

    public Circle(double radius, double xcoord, double ycoord, string co) //Constructor
    {
        r = radius;
        x = xcoord;
        y = ycoord;
        color = co;
    }

    public void draw()
    {
        Console.WriteLine("Circle with radius: {0} is drawn at: {1}:{2} in color: {3}",
            r, x, y, color);
    }

    public double Area()
    {
        return r*r*Math.PI;
    }
}

```

שימו לב שתכונות המעגל הוגדרו כמשתנים מסוג private. זה כדי להגן עליהם מפני שינוי מחוץ למחלקה - מה שניקרא: Data hiding או Data encapsulation

שימו לב שפעולת הציור היא רק מתודית ולא ממש ציור (כי לא למדנו גרפיקה בשפה).

בואו נעשה שימוש מיידי במחלקה שהגדרנו בדוגמה של תוכנית:

```
using System;
using System.Collections.Generic;
namespace Circles
{
    /* Here comes the definition of Circle as we outlined above */
    class MainClass
    {
        public static void Main(string[] args)
        {
            Circle c1= new Circle(4,2,3 "green"); //Instantiating one object
            Circle c2=new Circle(7,5,2, "red");    //Instantiating second object
            c1.draw();
            c2.draw();
            Console.WriteLine( c1.Area() );
        }
    }
}
```