

## מבוא לתכנות II

הרצאה 2 – מחלקה, אובייקט, בנאי והרשאות

גישה

סמסטר 2

# OOP – תכנות מונחה עצמים

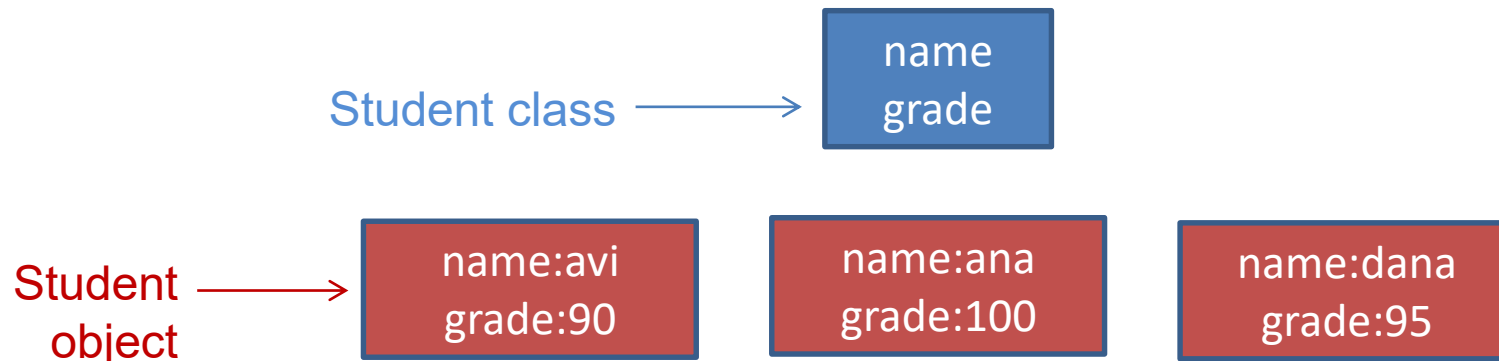
- OOP זו שיטת תכנות אשר כל דבר בה נבנה מתוך עצם (אובייקט). כמו בחיים האמיתיים, כמעט כל דבר הוא עצם (למעט דברים כמו נשמה, מחשבה...)
- בOOP אנו נוכל ליצור אובייקטים מסוג מחלקה מסוימת
- אנו קודם נגדיר מחלקה (מעין תבנית) על כל חלקיה ואז ניצור אובייקטים ממנה כמה שרק נרצה.
- דוגמה: מחלקת "סטודנט" יכולה להיות מוגדרת מ2 שדות: השם והציון
- שדות הם התכונות של המחלקה והם באים לידי ביטוי באמצעות משתנים.

Student class →

name  
grade

# OOP – תכנות מונחה עצמים

- אחרי שהגדרנו את מחלקת סטודנט נוכל ליצור כמה סטודנטים שרק נרצה. אלו ייקראו אובייקטים של סטודנט (מופעים). יש לתת לאובייקט (מופע) ערכים.
- נוכל ליצור, לדוגמה, 3 מופעי סטודנטים
- זהו היה הסבר "על קצה המזלג" של חשיבה של תכנות מונחה עצמים, כעת נעמיק בפרטים ובכתיבת קוד



# מחלקה

- כעת נגדיר את מחלקת הסטודנט. נעשה שימוש במילת המפתח *class*

– התעלמו מ *public* לעת עתה

– המחלקה יכולה להיות מוגדרת בתוך קובץ Program.cs כ"אחות" למחלקת Program (באותה היררכיה למשל מתחת למחלקת Program)

– או באמצעות Add → new Item → Class

```
class Student
{
    public string name;
    public int grade;
}
```

- שימו לב לכך שלא ניצלנו/תפסנו זיכרון עדיין, רק יצרנו Student

# אובייקט

- על מנת ליצור אובייקט חדש ממחלקת הסטודנט, נשתמש במילת המפתח *new*
- על מנת לגשת לשדות נשתמש ב '.' ובשם השדה

```
static void Main(string[] args)
{
    Student s1 = new Student();
    s1.name = "avi";
    s1.grade = 90;

    Student s2 = new Student();
    s2.name = "ana";
    s2.grade = 100;

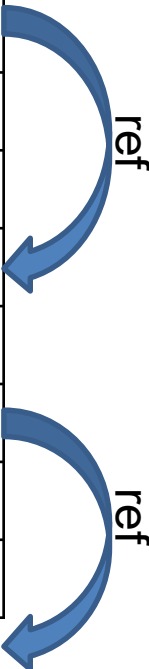
    Console.WriteLine(s1.name + " ; " + s1.grade);
    Console.WriteLine(s2.name + " ; " + s2.grade);
}
```

avi ; 90  
ana ; 100

# אובייקט והפנייה

- יצרנו שני אובייקטים של מחלקת סטודנט
- גם s1 וגם s2 הן הפניות לאובייקט של סטודנט.

address	Symbol	Value
#100	s1	#103
#101	b	3
#102		
#103	name	avi
#104	grade	90
#105	s2	#120
#106		
#107		



## = השמה באמצעות

- מה קורה בקוד הזה?

```
Student s1 = new Student();  
s1.name = "avi";  
s1.grade = 90;  
Student s2 = new Student();  
s2.name = "ana";  
s2.grade = 100;  
  
s1 = s2;  
s1.grade = 77;  
Console.WriteLine(s1.name + " ; " + s1.grade);  
Console.WriteLine(s2.name + " ; " + s2.grade);
```

?

ana ; 77

ana ; 77

- מה קרה לסטודנט avi? נוכל לגשת אליו שוב?

לא! אין לנו הפנייה אליו.

# סוגי נתונים

- יש שני סוגי של נתונים:

הפניה	ערך
string	int
array	float\double
class	boolean
	char

- סוגי "ערך" מכילים ערך יחיד ופשוט.
- סוגי "הפניה" מחזיקים נתונים מורכבים:
  - מחרוזת: רצף תווים
  - מערך: סדרה של ערכים אותו סוג רציפים בזכרון
  - מחלקה: סוג מותאם אישית שמאגד סוגי נתונים ומתודות ביחד
  - ערך ברירת המחדל (דיפולטיבי) להפניה הוא *null*



# מתודות

- בנוסף לשדות הפאסיביים שמתארים את המחלקה, אנחנו נרצה גם להעניק למחלקה פעולות שהיא מסוגלת לבצע. פעולות אלו נקראות מתודות (פונקציות).
- שדות ומתודות הם חברים המרכיבים מחלקה, ובעזרתם נוכל לתאר כל רעיון שנצטרך לממש בתוכנית ה C#.
- בואו ניתן לסטודנטים שלנו יכולות פעולה כגון האפשרות ללמוד כמה שעות. `void Learn(int hours)` נוכל להוסיף את השעות באופן ישיר לתוך הציון כך שנקבל קשר בין השעות שהסטודנט משקיע בלמידה והציון שלו. ככל שילמד יותר, כך ציונו ישתפר
- המתודות יכולות לגשת לכל השדות של המחלקה להן שייכות.

# מתודות

```
class Student
{
    //Fields
    public string name;
    public int grade;

    //Methods
    public void Learn(int hours)
    {
        grade += hours;
    }
}
```

```
Student s1 = new Student();
s1.name = "avi";
s1.grade = 90;
Console.WriteLine(s1.name + " ; " + s1.grade);
s1.Learn(5);
Console.WriteLine(s1.name + " ; " + s1.grade);
```

90

95

# מתודות

עוד מתודה שימושית היא מתודת `void Show()` - אשר מדפיסה את הפרטים של האובייקט

```
class Student
{
    //Fields
    public string name;
    public int grade;
    //Methods
    public void Learn(int hours)
    {
        grade += hours;
    }
    public void Show()
    {
        Console.WriteLine("my name is : " + name + " and my grade
is: " + grade);
    }
}
```

# מתודות

```
Student s1 = new Student();  
s1.name = "avi";  
s1.grade = 90;  
Student s2 = new Student();  
s2.name = "ana";  
s2.grade = 100;
```

my name is : avi and my grade is: 90

```
s1.Show();
```

```
s2.Show();
```

my name is : ana and my grade is: 100

# Constructor (ctor) = בנאי

- בכל פעם שאנו יוצרים אובייקט, הבנאי רץ ומאתחל את השדות לערכם הדיפולטיבי = ברירת מחדל

```
Student s1 = new Student();  
s1.Show();
```

my name is : and my grade is: 0

- נוכל לרשום *ctor* (בנאי) משלנו.
- יכול להיות יותר מ *ctor* אחד עם פרמטרים שונים.
- ה *ctor* הדיפולטיבי (ברירת מחדל, הריק) קיים רק אם לא יצרנו אחד אחר.
- ה *ctor* תמיד יופיע בצורה של `public ClassName (...) {...}`. אין סוג החזרה.

- כאן יש לנו 3 בנאים שונים. הבנאי הדיפולטיבי לא קיים עוד.

```
...
//Methods
public Student()
{
    name = "no name";
    grade = 80;
}

public Student(int g)
{
    name = "no name";
    grade = g;
}

public Student(string n, int g)
{
    name = n;
    grade = g;
}
...
```

- השימוש בבנאי נעשה בזמן יצירת האובייקט, לאחר השימוש במילת המפתח *new*

```
...  
Student s1 = new Student();  
s1.Show();  
Student s2 = new Student(99);  
s2.Show();  
Student s3 = new Student("sarit", 100);  
s3.Show();  
...
```

my name is : no name and my grade is: 80

my name is : no name and my grade is: 99

my name is : sarit and my grade is: 100

# הרשאות גישה

- ישנן 5 הרשאות גישה:

Public, private, protected, internal, internal protected –

– בהתאמה: ציבורי, פרטי, מוגן, פנימי, פנימי מוגן

– אנו נלמד את "פרטי" ו"ציבורי" בנקודה זו בקורס.

- "פרטי" הכוונה שהמשתנה נגיש וזמין אך ורק בתוך המחלקה לה הוא שייך, ולא דרך אף מחלקה אחרת (מחוץ למחלקה).

- "ציבורי" הכוונה שהמשתנה נגיש וזמין מכל מקום (גם מחוץ למחלקה שלו)



# הרשאות גישה

## • דוגמה:

```
class Student
{
    //Fields
    public string name;
    private int grade;
    ...
    public void Learn(int hours)
    {
        grade += hours;
    }
}
```

נגיש וזמין

```
class Program
{
    ...
    Student s1 = new Student();
    s1.name = "avi";
    s1.grade = 100; //ERROR - does not compile
    ...
}
```

לא נגיש או זמין

# מילת מפתח: *this*

- מילת המפתח *this* היא הפניה אל האובייקט הנוכחי ושימושה במניעת בלבול בין שמות שדות המחלקה והפרמטרים של הבנאי.

```
class Student
{
...
public Student(string name, int grade)
{
    name = name; //both are the local variable name!
    grade = grade; //both are the local variable grade!
}
```

```
class Program
{
...
Student s3 = new Student("sarit", 100);
s3.Show();
```

my name is : and my grade is: 0

# this

- כעת נשתמש במילת המפתח *this* כהפניה לאובייקט הנוכחי


```
class Student
{
...
public Student(string name, int grade)
{
    this.name = name;
    this.grade = grade;
}
```

```
class Program
{
...
Student s3 = new Student("sarit", 100);
s3.Show();
```

my name is : sarit and my grade is: 100

# this

- שימוש נוסף למילה *this* הוא בשרשור בנאים – קריאה לבנאי אחד מבנאי אחר



```
...  
public Student(int g)  
{  
    name = "no name";  
    grade = g;  
}  
  
public Student(string name, int grade) :this(grade)  
{  
    this.name = name;  
}  
  
...
```

# מערך של אובייקטים

- כיצד ניתן ליצור כיתה של סטודנטים?
- ניצור מערך של סטודנטים!

```
Student[] CS_Class = new Student[3];  
Student s1= new Student("avi", 90);  
CS_Class[0] = s1;  
CS_Class[1] = new Student("sarit", 100);  
CS_Class[2] = new Student(80);  
CS_Class[2].name = "dudu";  
CS_Class[0].Show();  
CS_Class[1].Show();  
CS_Class[2].Show();
```

my name is : avi and my grade is: 90  
my name is : sarit and my grade is: 100  
my name is : dudu and my grade is: 80

# מערך של אובייקטים

## • שימוש ב FOR

```
Student[] CS_Class = new Student[3];
for (int i = 0; i < CS_Class.Length; i++)
{
    CS_Class[i] = new Student(Console.ReadLine(),
int.Parse(Console.ReadLine()));
}
for (int i = 0; i < CS_Class.Length; i++)
{
    CS_Class[i].Show();
}
```

```
avi
90
sarit
100
dudu
80
my name is : avi and my grade is: 90
my name is : sarit and my grade is: 100
my name is : dudu and my grade is: 80
```

# כל דבר הוא מחלקה

- כל קטע קוד נמצא בתוך פונקציה
- כל פונקציה נמצאת בתוך מחלקה
- מחלקה היא ההגדרה עבור אובייקט
- כל מה שנקודד יהיה מחלקות
- מופע הוא אובייקט של מחלקה
- מחלקה יכולה להכיל שדות = משתנים ומתודות = פונקציות

- ממשו את מחלקת "סטודנט" שלפניכם:

```
class Student
{
    //Fields
    private int Sid ;
    private int Sgrade ;
    private string Sname ;

    public void SetID(int newId )...

    public void SetGrade(int newGrade )...

    public void SetName(string newName )...

    public void PrintStudent()...
}
```



## • דוגמה נוספת carAndPerson

```
class Program...
{
    class Person
    {
        string name;
        int age;
        Car myCar;

        public Person()...

        public void GetCar(Car c)...

        public Person(string n , int a, Car c)...

        public void Print()...
    }
}
```

```
class Car
{
    private string model;
    private int speed;
    private double engine;
    private bool autoGear;

    public Car()...

    public Car(string model, int speed, double engine, bool AG)...

    public void Print( )...

    public void SetSpeed(int speed)...

    public int GetSpeed()...

    public void SetGear(bool g)...

    public bool GetGear()...
```

## • דוגמה של וקטור

```
public class IntegersVector
{
    /// <summary>
    /// holds the integers.
    /// </summary>
    private int[] integers;

    /// <summary>
    /// the position of the last integer inserted.
    /// </summary>
    private int position;

    /// <summary>
    /// Creates a new integers vector.
    /// </summary>
    public IntegersVector()...

    /// <summary>
    /// Appends the specified element to the end of this Vector.
    /// </summary>
    /// <param name="newInteger">The integer to add.</param>
    public void Add(int newInteger)...

    /// <summary>
    /// Returns the element at the specified position in this Vector.
    /// </summary>
    /// <param name="index">the index of the int to return.</param>
    public int Get(int index)...
```

```
/// <summary>
/// Tests if this vector has no integers.
/// </summary>
/// <returns>True is there are no integers in the
/// Vector and false otherwise.</returns>
public bool IsEmpty()...

/// <summary>
/// Returns the number of integers in this vector.
/// </summary>
/// <returns>The number of integers in the vector.</returns>
public int Size()...

/// <summary>
/// Doubles the vector size.
/// </summary>
private void EnlargeVector()...
```