

תכנות מונחה עצמים

Object Oriented Programming

תכנות מונחה עצמים

Object Oriented Programming

גישה ישנה:

Procedural Programming – תוכנית ארוכה שמחולקת לאוסף של פונקציות.

גישה חדשה:

Object Oriented Programming – גישה שאמורה להתאים לדרך החשיבה שלנו כבני אדם (עבודה עם דברים יותר מוחשיים).

תכנות מונחה עצמים

Object Oriented Programming

אבני היסוד:

(1) מחלקה

(2) אובייקט

מחלקה Class

מבנה לוגי שמאגד בתוכו משתנים ו/או פונקציות.

משתני המחלקה - נקראים מאפיינים או תכונות (attributes או members).

פונקציות המחלקה - נקראות מתודות או שיטות (operations או methods).

אובייקט Object

אובייקט = עצם = מופע = Object = Instance

אובייקט = משתנה מסוג המחלקה

```
int num;  
Person p1;  
Worker w1;  
Student s1;  
Dog d1;
```

המחלקה Worker - תרשימים UML בסיסי

```
+-----+
|                Worker                |
+-----+
| + id : int                               |
| + name : string                         |
| + workingHours : int                   |
| + salaryPerHour : double               |
+-----+
| + CalculateSalary() : double           |
| + Print() : void                       |
+-----+
```

יצירת מחלקה



גוף המחלקה מכיל מאפיינים ו/או מתודות.

```
class Worker
{
    ...
}
```

יצירת אובייקט

```
Worker w1 = new Worker();
```

w1 = 2000

שם = ""

ת.ז. = ""

שעות עבודה = 0

תעריף שעתי = 0

השמת ערך למאפיין והפעלת מתודה

```
Worker w1 = new Worker();
```

```
w1.name = "Shaked";
```

```
w1.Print();
```

w1 = 2000

"Shaked" = שם

"" = ת.ז.

0 = שעות עבודה

0 = תעריף שעתי

המחלקה Worker - בואו נשדרג אותה

בואו נוסיף:

הרשאות גישה

Setters

Getters

בנאים

הרשאות גישה למאפיינים ומתודות

Access Modifiers

1. הרשאת גישה נועדה לצורך הסתרת מידע Information Hiding (בדרך כלל מאפיינים) ומתן גישה לחלקים שרוצים לחשוף בתוכנית (בדרך כלל מתודות).

2. מי שקובע מי יכול לגשת למאפיין או מתודה זאת המחלקה ולא האובייקט.

הרשאות גישה למאפיינים ומתודות

3. ישנן חמש רמות הרשאה (נכיר בהתחלה את השניים הראשונים):

א. **private** (פרטי) – מותרת הגישה מתוך המחלקה הנוכחית בלבד (זאת גם הרשאת ברירת מחדל של המאפיינים והמתודות). בד"כ נגדיר מאפיינים כפרטיים.

ב. **public** (ציבורי) – מותרת הגישה מכל מקום. בד"כ נגדיר מתודות שרוצים לחשוף החוצה כציבוריות.

הרשאות גישה אחרות (נכיר בהמשך):

ג. **protected** (מוגן)

ד. **internal** (פנימי)

ה. **internal protected** (פנימי מוגן)

מתודות Set ו- Get

מתודות שנגדיר כ- public.

בד"כ מקובל לא לתת גישה ישירה למאפייני המחלקה (נגדיר אותם כ- private למשל) והגישה אליהם תהיה ע"י מתודות Set ו- Get.

מתודת Set – נקראת גם Setter method – תפקידה לתת אפשרות לעדכון המאפיין.

מתודת Get – נקראת גם Getter method – תפקידה להחזיר את המאפיין למי שביקש אותו.

בתוך מתודות Set ו- Get נוכל להוסיף לוגיקה שבודקת מי זה שמבקש לבצע את השינוי ולפי זה נחליט האם נרצה לבצע את השינוי או לא.

מתודות Set ו-Get - דוגמא

```
class Worker
{
    private string name;

    public void SetName(string name)
    {
        this.name = name;
    }

    public string GetName()
    {
        return this.name;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Worker w1 = new Worker();
        w1.name = "Sean"; //is this ok?
        w1.SetName("Sean"); //is this ok?
        Console.WriteLine(w1.GetName());
    }
}
```

בנאי – מתודת בנייה - constructor

תפקיד הבנאי הוא אתחול האובייקט בזמן הגדרתו (כדי שהאובייקט יקבל ערכים אמיתיים).

הבנאי נקרא לאחר שהזיכרון עבור האובייקט כבר הוקצה.

בנאי יכול לקבל ערכים אך אינו מחזיר ערכים (אפילו לא void).

קיים בנאי ברירת מחדל של השפה שתפקידו לשים ערכי ברירת מחדל עבור המאפיינים.

אם בונים בנאי משלנו אנחנו דורסים override את בנאי ברירת המחדל של השפה.

ניתן לעשות העמסת בנאים constructor overloading.

שם הבנאי כשם המחלקה.

סוגי בנאים

בנאי ברירת מחדל

בנאי שמקבל פרמטרים

בנאי מעתיק

בנאי ברירת מחדל

לא מקבל ערכים ולא מחזיר ערכים.

קיים בשפה בנאי ברירת מחדל שמאתחל את הערכים בערכי ברירת מחדל.

אם בונים בנאי ברירת מחדל משלנו אז יידרס בנאי ברירת המחדל של השפה.

דוגמא:

```
public Worker()
{
    this.name = "No Name";
    this.id = "000";
    this.workingHours = 100;
    this.salaryPerHour = 50;
}

class Program
{
    static void Main(string[] args)
    {
        Worker w1 = new Worker();
    }
}
```

בנאי שמקבל פרמטרים

מקבל ערכים ולא מחזיר ערכים.

אם בונים בנאי משלנו אז יידרס בנאי ברירת המחדל של השפה.

כשבונים אובייקט נהיה חייבים לשלוח את הפרמטרים שציינו בזמן הגדרת האובייקט.

דוגמא:

```
public Worker(string name, string id, double workingHours, double salaryPerHour)
{
    this.name = name;
    this.id = id;
    this.workingHours = workingHours;
    this.salaryPerHour = salaryPerHour;
}

class Program
{
    static void Main(string[] args)
    {
        Worker w2 = new Worker("Sean", "042842154", 150, 60);
    }
}
```

בנאי מעתיק copy constructor

מקבל אובייקט קיים ולא מחזיר ערכים.

דומה לבנאי עם פרמטרים אך מקבל אובייקט קיים.

תפקידו להעתיק את תוכן האובייקט ששלחנו לאובייקט החדש.

אם בונים בנאי משלנו אז יידרס בנאי ברירת המחדל של השפה.

דוגמא:

```
public Worker(Worker obj)
{
    this.name = obj.name;
    this.id = obj.id;
    this.workingHours = obj.workingHours;
    this.salaryPerHour = obj.salaryPerHour;
}

class Program
{
    static void Main(string[] args)
    {
        Worker w2 = new Worker("Sean", "042842154", 150, 60);
        Worker w3 = new Worker(w2);
    }
}
```

Constructor overloading

ניתן לעשות העמסת בנאים (בניית כמה בנאים שונים עם מספר פרמטרים שונה ו/או טיפוס פרמטרים שונה).

יופעל הבנאי המתאים לפי הפרמטרים ששלחנו כשבנינו את האובייקט.

```
public Worker()
{
    this.name = "No Name";
    this.id = "000";
    this.workingHours = 100;
    this.salaryPerHour = 50;
}

public Worker(string name, string id, double workingHours, double salaryPerHour)
{
    this.name = name;
    this.id = id;
    this.workingHours = workingHours;
    this.salaryPerHour = salaryPerHour;
}

public Worker(Worker obj)
{
    this.name = obj.name;
    this.id = obj.id;
    this.workingHours = obj.workingHours;
    this.salaryPerHour = obj.salaryPerHour;
}

class Program
{
    static void Main(string[] args)
    {
        Worker w1 = new Worker();

        Worker w2 = new Worker("Sean", "042842154", 150, 60);

        Worker w3 = new Worker(w2);
    }
}
```

המחלקה Worker - תרשים UML מתקדם

```
+-----+
|                               |
|                               | Worker                               |
|                               |
+-----+
| - id : int                   |
| - name : string              |
| - workingHours : int         |
| - salaryPerHour : double     |
|                               |
+-----+
| + Worker()                  |
| + Worker(id:int, name:string, |
|           workingHours:int, salaryPerHour:double) |
| + Worker(other:Worker)      |
| + GetId() : int             |
| + SetId(id:int) : void       |
| + GetName() : string         |
| + SetName(name:string) : void |
| + GetWorkingHours() : int     |
| + SetWorkingHours(h:int) : void |
| + GetSalaryPerHour() : double |
| + SetSalaryPerHour(s:double) : void |
| + CalculateSalary() : double |
| + Print() : void             |
+-----+
```

תרגיל בית - המחלקה Worker - תרשים UML מתקדם

1. עליך לממש את המחלקה Worker לפי תרשים UML המוצג.
2. בפונקציה Main עליך ליצור שלושה אובייקטים מסוג המחלקה Worker כאשר כל עובד עליך לבנות בעזרת בנאי אחר.
3. קרא לפונקציות Get/Set המתאימות.
4. קרא לפונקציה Print להדפסת העובדים.