# *סיכום כללי – oop*

## הגדרות מונחים:

**מחלקה** – **תבנית** ליצירת מופעים, המגדירה **טיפוס** שמכיל משתנים ופעולות.

ממחלקה אפשר ליצור מופעים על ידי המילה new והמופעים הנ"ל יווצרו בheap (יהיה להם מצביע)

**אובייקט** – מופע בודד של מחלקה (נוצר על ידי הבנאי)

**Class members** – פונקציות (פעולות) (בנאי הוא גם פונקציה) שמוגדרות בתוך המחלקה, ומשתנים (מאפיינים) שמוגדרים בתוך המחלקה

**הרשאת גישה** – הרשאות גישה יכולות להיות public / private ועוד, הגדרת המקומות מהם נוכל לגשת לclass member

**private** – הרשאת גישה , המגדירה את ההרשאה לגשת אל ה class member רק מתוך המחלקה הספציפית (לא דרך האובייקט, ולא דרך הנגזרת) – אם ניגש מחוץ למחלקה נקבל שגיאת קומפילציה

**public** - הרשאת גישה , המגדירה את ההרשאה לגשת אל ה class member מכל מקום (גם דרך המחלקה עצמה, גם דרך האובייקט, וגם דרך הנגזרת)

**static** – הגדרה ברמת המחלקה ולא ברמת האובייקט (המופע) הבודד, מאפשר לגשת ישירות לclass member הסטטי, ללא צורך ביצירת מופע

- מתוך המחלקה ניגש לclass member הסטטי דרך class name ולא דרך this
- ממחלקה יורשת ניגש לclass member הסטטי של הבסיס דרך class name ולא דרך this או super
- מחוץ המחלקה ניגש לclass member הסטטי דרך class name ולא דרך האובייקט

**Ctor –** בנאי – פונקציה דיפולטיבית הנמצאת בצורה אוטומטית בכל מחלקה שניצור.

על ידי האופרטור new נוכל לגשת לבנאי וליצור בheap אובייקט חדש מטיפוס המחלקה.

הבנאי יוצר בפועל מופע בזיכרון ומחזיר את הכתובת של המופע שיצר.

אם נגדיר בנאי שמקבל ערכים נהיה חייבים לשלוח את הפרמטרים בקריאה של הnew

**new –** אופרטור המשמש לקריאה לבנאי של מחלקה, לשם יצירת אובייקט מהמחלקה

**Composition** – הכלה של אובייקט ממחלקה A בתוך מחלקה B

**Inheritance** הורשה של מחלקה A למחלקה B (מחלקה B תמיד תכיל בתוכה את כל ה class members של מחלקת הבסיס)

- מחלקה A - מחלקת בסיס
- מחלקה B - מחלקת נגזרת
- יצירת אובייקט ממחלקה B - תקרא לבנאי של B ומשם בצורה מרומזת / מפורשת (ע"י super()) לבנאי של המחלקה A
- מהמחלקה הנגזרת אי אפשר לגשת לclass member המוגדרים כprivate בבסיס

**Multi level inheritance** – מחלקה c יורשת ממחלקה B שיורשת ממחלקה A (מחלקה C תמיד תכיל בתוכה את כל הclass members של A + B)

```
class A { }
class B:A { }
class C:B { }
```

**Multi inheritance**—קבלת ירושה במחלקה C ממחלקה A וממחלקה B – דבר זה לא אפשרי ב #c (וברוב שפות התכנות האחרות)

```
class A { }
class B { }
class C:B,A { } //אסור!
```

**Override** – דריסה של פונקציה ממחלקת הבסיס במחלקה הנגזרת. לשם כך יש לענות על הדרישות הבאות:

- הפונקציה בבסיס חייבת להיות עם אותו שם כמו הפונקציה בנגזרת
- הפונקציה בבסיס חייבת לקבל את אותו מספר פרמטרים שבפונקציה הדורסת (והפרמטרים חייבים להיות בעלי טיפוס זהה לפונקציה הדורסת)
- הפונקציה בבסיס חייבת להחזיר את אותו טיפוס שמוחזר מהפונקציה הדורסת

**Abstract** - מחלקה מופשטת (מחלקה בסיסית וראשונית ממנה נירש למחלקות ממשיות יותר), ממחלקה אבסרקטית אי אפשר ליצור מופעים, אבל אפשר להגדיר משתנה שיהיה מטיפוס המחלקה האבסרקטית ויוכל להצביע לכל אובייקט נגזרת (פולימורפיזם).

**Interface** – ממשק – מגדיר מעין "חוזה" שיכול להיות ממומש על ידי מחלקות

**is-** אופרטור המשמש לבדיקה:

- האם אובייקט מסוים הוא מופע של מחלקה מסוימת, או נגזרת של אותה מחלקה.

האופרטור יקבל בצד שמאל את שם האובייקט, ובצד ימין את שם המחלקה שנרצה לבדוק.

- האם אובייקט מסוים הוא מופע של מחלקה שמממשת ממשק מסוים

האופרטור יקבל בצד שמאל את שם האובייקט, ובצד ימין את שם הממשק שנרצה לבדוק.

**down casting** – המרה של אובייקט למחלקה אחרת (בד"כ ממחלקה בסיסית יותר למחלקה נגזרת)

```
class A { }
class B:A { }
```

```
A obj = new B();
B b1=(B)obj;        // down cast - way 1
B b2 = obj as B;    // down cast - way 2
```

**ToString** – פונקציה שמוגדרת למחלקה object – ב#c כל מחלקה יורשת בצורה אוטומטית מהמחלקה object

בתוך המחלקה object מוגדרת הפונקציה הבאה:

```
//
// Summary:
//     Returns a string that represents the current object.
//
// Returns:
//     A string that represents the current object.
public virtual string ToString();
```

- אם נדרוס את הפונקציה בנגזרת – נקבל את תוצאת הדריסה
- אם לא נדרוס את הפונקציה בנגזרת – נקבל את שם המחלקה (זהו הפלט הדיפולטיבי של הפונקציה)

```
using System;


namespace app
{
    class A { }

    class B {
        public override string ToString()
        {
            return "I am class B";
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            A objA = new A();
            B objB = new B();

            Console.WriteLine(objA.ToString());  // --> app.A
            Console.WriteLine(objB.ToString());  // --> I am class B

        }
    }
}
```

# סיכום הרשאות והורשות:

| | מחלקה רגילה | מחלקה אבסטרקטית | ממשק |
|---|---|---|---|
| **הגדרת משתנה private** | כן | כן | לא |
| **הגדרת משתנה public** | כן | כן | לא |
| **הגדרת משתנה ללא הרשאת גישה** | כן | כן | לא |
| **הגדרת בנאי** | כן | כן | לא |
| **הגדרת פונקציה public כולל מימוש** | כן | כן | לא |
| **הגדרת פונקציה private כולל מימוש** | כן | כן | לא |
| **הגדרת פונקציה abstract private** | לא | לא | לא |
| **הגדרת פונקציה abstract public** | לא | כן | לא |
| **הגדרת פונקציה ללא מימוש וללא המילה abstract** | לא | לא | כן |
| **ירושה של מחלקה רגילה (extends)** | כן<br>• נוכל לגשת למשתנים לא סטטיים public של הבסיס על ידי המילה this<br>• נוכל לגשת למשתנים סטטיים public של הבסיס על ידי שם מחלקת הבסיס<br>• נוכל לגשת לפונקציות סטטיות public של הבסיס על ידי שם מחלקת הבסיס<br>• נוכל לגשת לפונקציות לא סטטיות public של הבסיס על ידי המילה this או base – אם דרסנו את פונקציית הבסיס נוכל לפנות רק על ידי base | כן | לא |
| **הגדרת משתנה (מצביע)** | כן – יכול להצביע על אובייקטים מהסוג שלו או על אובייקטי נגזרת | כן – יכול להצביע רק על אובייקטי נגזרת | כן – יכול להצביע רק על אובייקטי מחלקה שמממשת את הממשק |
| **יצירת אובייקט (על ידי new)** | כן | לא | לא |
| **גישה על ידי מצביע בסיס למשתני נגזרת (שלא קיימים בבסיס)** | לא – חייב המרה מקומית לטיפוס הנגזרת | לא – חייב המרה מקומית לטיפוס הנגזרת | לא – חייב המרה מקומית לטיפוס הנגזרת |

# Accessibility Levels

| Declared accessibility | Meaning |
|---|---|
| `public` | Access is not restricted. |
| `protected` | Access is limited to the containing class or types derived from the containing class. |
| `internal` | Access is limited to the current assembly. |
| `protected internal` | Access is limited to the current assembly or types derived from the containing class. |
| `private` | Access is limited to the containing type. |
| `private protected` | Access is limited to the containing class or types derived from the containing class within the current assembly. |

## Access modifiers on namespaces

Access modifiers are not allowed on namespaces. Namespaces have no access restrictions.

## Access modifiers on Top-level types

Top-level types, which are not nested in other types, can only have `internal` or `public` accessibility. The default accessibility for these types is `internal`.

## Access modifiers on Nested types

Nested types, which are members of other types, can have declared accessibilities as indicated in the following table.

| Members of | Default member accessibility | Allowed declared accessibility of the member |
|---|---|---|
| `enum` | `public` | None |
| `class` | `private` | `public`<br>`protected`<br>`internal`<br>`private`<br>`protected internal`<br>`private protected` |
| `interface` | `public` | None |