OOP-C#

Object-Oriented Programming

Lesson 3

©איילת ויץ 2023

הורשה-המשך

דגשים חשובים לגבי הורשה:

כל מה שיש במחלקה עובר בהורשה כולל: משתנים, תכונות, פונקציות וכו'.**למעט בנאים.**

אלמנטים שמוגדרים כprivate עוברים בהורשה אך לא ניתן לגשת אליהם מהמחלקה היורשת.

השימוש במחלקה היורשת הוא כשימוש במחלקה רגילה.

במחלקה היורשת חייב להפעיל בנאי כלשהו ממחלקת הבסיס

הבנאי של מחלקת הבסיס יופעל קודם לבנאי של המחלקה היורשת.

אם לא נציין במפורש איזה בנאי נרצה להפעיל יופעל אוטומטית הבנאי שאינו מקבל פרמטרים.

©איילת ויץ 2023

הפעלת בנאי ממחלקת הבסיס מתבצעת בעזרת המילה השמורה :base

```
internal class Animal
{
        private String name;
        private int numLegs;

        public Animal(String name,int numLegs)
{
            this.name = name;
            this.numLegs = numLegs;
}

}

internal class Fish:Animal
{
        public Fish(String name,int numLegs):base(name,numLegs){ }

}
```

©איילת ויץ 2023

שימוש בפונקציה במחלקת הבסיס

לעיתים קיימת פונקציה במחלקת הבסיס אך היא אינה מתאימה בדיוק במימוש שלה למחלקה היורשת. לדוגמא: פונקציה שמדפיסה את כל המאפיינים של המחלקה. לעיתים היא לא מתאימה כלל ולעיתים היא מתאימה חלקית.

ב- C# ניתן לדרוס פונקציה שקיבלנו ממחלקת הבסיס ואם נרצה (לא חובה) נוכל להשתמש בתוך המחלקה היורשת גם בפונקציה שקיבלנו ממחלקת הבסיס. השימוש יתבצע באמצעות המילה השמורה **base** ולאחריה נקודה וקריאה לפונקציה.

```
 public override void Iam()
{
            base.Iam();
            Console.WriteLine("I am Fish");
}

public override String ToString()
{
            return base.ToString() + " isFins: " + isFins;
}
```

©איילת ויץ 2023

פולימורפיזם
פולי – מורפיזם = רב – צורתיות.

פולימורפיזם הוא אחד הכלים החזקים ביותר בתכנות מונחה עצמים, ומשמעותו העיקרית היא:
המשותף הוא הממשק ולא המימוש – כלומר: ממשק משותף, מימוש נפרד.

זהו מנגנון להתייחסות לעצמים מטיפוסים שונים באופן אחיד.
ניתן להתייחס לכל עצם כאילו היה מופע של כל אחת מהמחלקות מהן הוא יורש.
כל משתנה הפניה של מחלקת בסיס יכול להצביע בפועל על עצם ממחלקה נגזרת.

שימושים של פולימורפיזם
יצירת מערך הטרוגני – ניתן ליצור מערך מסוג מחלקת הבסיס כך שכל איבר במערך יהיה אובייקט מסוג מחלקה יורשת-נגזרת ובאשר נרוץ על המערך ונפעיל את הפונקציה, תופעל הפונקציה המתאימה לאובייקט. היתרון הוא שלא נצטרך לבדוק עבור כל איבר מה הטיפוס שלו.

קבלה לפונקציה – ניתן לבנות פונקציה המקבלת משתנה מסוג מחלקת הבסיס ולשלוח אליה בכל פעם אובייקט מסוג אחר היורש ממחלקת הבסיס.

©איילת ויץ 2023

מימוש Polymorphism
בבדי לממש פולימורפיזם יש תחילה להגדיר את הפונקציה במחלקת הבסיס **כוירטואלית** (virtual)
מתודה המוגדרת כ **virtual באבא** מאפשרת למחלקת הבן לממש אותה.
הבן יוכל לממש את המתודה הזו ב 2 צורות:
1. override-דריסה
במחלקה היורשת נדרוס את הפונקציה באמצעות **override**.

אם העצם הוא מטיפוס הבן ממש, אז כשהוא יפעיל את המתודה, תיקרא המתודה של הבן ולא של האבא.
גם אם ההפניה (המצביע) הוא מטיפוס האבא ובפועל הוא מצביע לעצם מטיפוס הבן, גם אז, תיקרא המתודה **של הבן ולא של האבא.**

בנוסף חשוב לדעת שפונקציה שביצענו ל- override היא גם וירטואלית לדור הבא שירש אותה.

2. New – הפסקת הדריסה
אם העצם הוא מטיפוס הבן ממש, אז כשהוא יפעיל את המתודה, תיקרא המתודה של הבן ולא של האבא.
אם ההפניה (המצביע) הוא מטיפוס האבא ובפועל הוא מצביע לעצם מטיפוס הבן, תיקרא המתודה **הוירטואלית של האבא הקרוב ביותר ולא של הבן.**

©איילת ויץ 2023

במחלקת Animal (מחלקת האב) נגדיר את הפונקציה כווירטואלית:

```
public virtual void Iam()
{
            Console.WriteLine("I am animal");
}
```

במחלקת Fish

1. באמצעות override:

```
 public override void Iam()
{
            Console.WriteLine("I am Fish");
}
```

2. באמצעות new:

```
 public new void Iam()
{
            Console.WriteLine("I am Fish");
}
```

©איילת ויץ 2023

הערות חשובות:

ברירת המחדל בהגדרת מתודה היא new, כלומר אם לא מוגדר שום דבר, המתודה לא תדרוס את המתודה של האב.

לא ניתן לבצע override אם המתודה במחלקת האב לא הוגדרה כ virtual.

מתודה שהוגדרה כ override נחשבת כווירטואלית ליורשים ממנה.

©איילת ויץ 2023

המרות בין אבא לבן

המרה כלפי מעלה ניתן לעשות עם המרה מרומזת

```
Animal a = new Fish();
```

המרה כלפי מטה, הקומפיילר מחייב המרה מפורשת

```
Fish ff = (Fish)a;
```

דוגמא נוספת:

```
Animal a2 = new Fish("Forel", 0, true);
```

גישה לאובייקט

כאשר הייחוס הוא מסוג הבסיס והאובייקט הוא מסוג יורש לא ניתן לגשת לאלמנטים של היורש, אם ננסה לעשות זאת תתקבל שגיאת קומפילציה. אם בכל זאת נרצה לעשות נצטרך להצביע על האובייקט עם ייחוס המתאים לאובייקט (מסוג היורש).
לדו':

```
Console.WriteLine(a2.IsFins);
```
שגיאה!

ולכן על מנת לגשת לתכונות של היורש, נצטרך לבצע המרה מקומית.

```
Console.WriteLine(((Fish)a2).IsFins);
```

©איילת ויץ 2023

מחלקה אבסטרקטית

מחלקה אבסטרקטית הינה מחלקה אשר מהווה בסיס למחלקות אחרות, אך אין לנו באמת צורך באובייקטים ממנה בתוכנית.
כאשר יוצרים מחלקה אבסטרקטית, לא ניתן ליצור ממנה מופעים אלא רק מהמחלקות היורשות ממנה.

- בבדי ליצור מחלקה אבסטרקטית יש לכתוב **abstract** לפני שם ה- class
- לא ניתן ליצור אובייקטים ממחלקה אבסטרקטית
- ניתן ליצור ייחוס ממחלקה אבסטרקטית (בד"כ לצורכי פולימורפיזם)
- מחלקה אבסטרקטית יכולה להכיל כל דבר שמחלקה רגילה יכולה להכיל
- מחלקה אבסטרקטית יכולה (אבל לא חייבת) להכיל פונקציות אבסטרקטיות
- לא ניתן לכתוב פונקציה אבסטרקטית במחלקה שהיא לא אבסטרקטית
- פונקציה אבסטרקטית היא פונקציה שאין לה מימוש. היא נכתבת כהצהרה ללא גוף (כלומר ללא בלוק של פקודות)
- כאשר יורשים מחלקה אבסטרקטית חובה לממש את כל הפונקציות האבסטרקטית שבה (אם יש)
- המימוש של פונקציה אבסטרקטית הוא באמצעות מילת המפתח **override**
- אין לכתוב מימוש {...} של מתודה אבסטרקטית. אם זהו מאפיין, נסתפק ב set/get

©איילת ויץ 2023

הבדלים בין פונקציה virtual לפונקציה abstract

	abstract	virtual
מימוש המתודה במחלקת האב	אין מימוש של המתודה - לא צריך להגדיר את גוף המתודה, רק כותרת.	יש לממש את המתודה (בדומה למתודה רגילה). משמש לפולימורפיזם.
מימוש המתודה במחלקות היורשות	חובה לממש במחלקות היורשות בצורה ישירה. אחרת הן בעצמן מופשטות.	לא חובה לממש.
מיקום הגדרת המתודה	אפשר להגדיר מתודה אבסטרקטית רק במחלקה אבסטרקטית	בכל מחלקה
הרשאת גישה	**לא** private (משום שאז המתודה לא תיראה במחלקות הנגזרות ולא נוכל לממש אותה)	

©איילת ויץ 2023

דוגמא למחלקה אבסטרקטית:

```
 abstract class Shape
}
        public abstract string Iam();
{
```

מחלקה היורשת ממחלקה אבסטרקטית:

```
 class Circle : Shape
{
    public override String Iam()
{
            return "I am Circle";
}
}
```

©איילת ויץ 2023

מחלקה אבסטרקטית

מחלקה אבסטרקטית הינה מחלקה אשר מהווה בסיס למחלקות אחרות, אך אין לנו באמת צורך באובייקטים ממנה בתוכנית.
כאשר יוצרים מחלקה אבסטרקטית, לא ניתן ליצור ממנה מופעים אלא רק מהמחלקות היורשות ממנה.

- בבדי ליצור מחלקה אבסטרקטית יש לכתוב **abstract** לפני שם ה- class
- לא ניתן ליצור אובייקטים ממחלקה אבסטרקטית
- ניתן ליצור ייחוס ממחלקה אבסטרקטית (בד"כ לצורכי פולימורפיזם)
- מחלקה אבסטרקטית יכולה להכיל כל דבר שמחלקה רגילה יכולה להכיל
- מחלקה אבסטרקטית יכולה (אבל לא חייבת) להכיל פונקציות אבסטרקטיות
- לא ניתן לכתוב פונקציה אבסטרקטית במחלקה שהיא לא אבסטרקטית
- פונקציה אבסטרקטית היא פונקציה שאין לה מימוש. היא נכתבת כהצהרה ללא גוף (כלומר ללא בלוק של פקודות)
- כאשר יורשים מחלקה אבסטרקטית חובה לממש את כל הפונקציות האבסטרקטית שבה (אם יש)
- המימוש של פונקציה אבסטרקטית הוא באמצעות מילת המפתח **override**
- אין לכתוב מימוש {...} של מתודה אבסטרקטית. אם זהו מאפיין, נסתפק ב set/get

©איילת ויץ 2023

תרשימי UML

מחלקה תיוצג ע"י מלבן, לפי הסדר הבא:

שם המחלקה

שמות השדות

שם

טיפוס

אתחול, אם ישנו

שמות המתודות

פרמטרים וסוגיהם

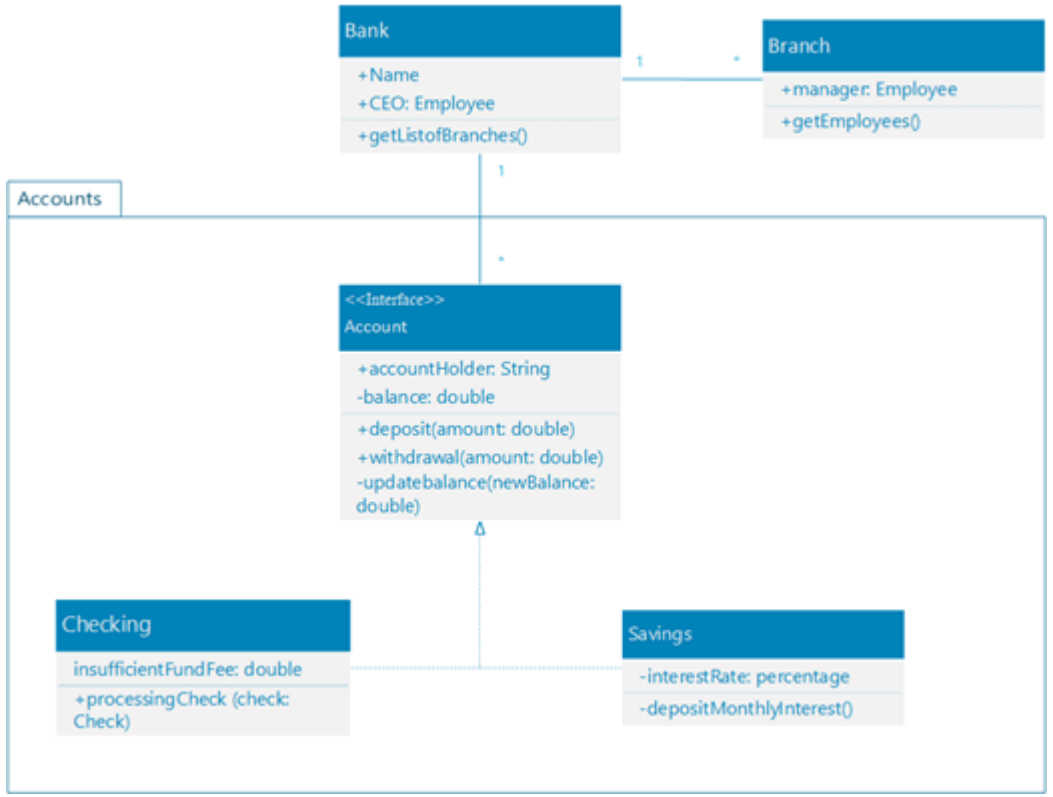
ערך מוחזר, אם ישנו

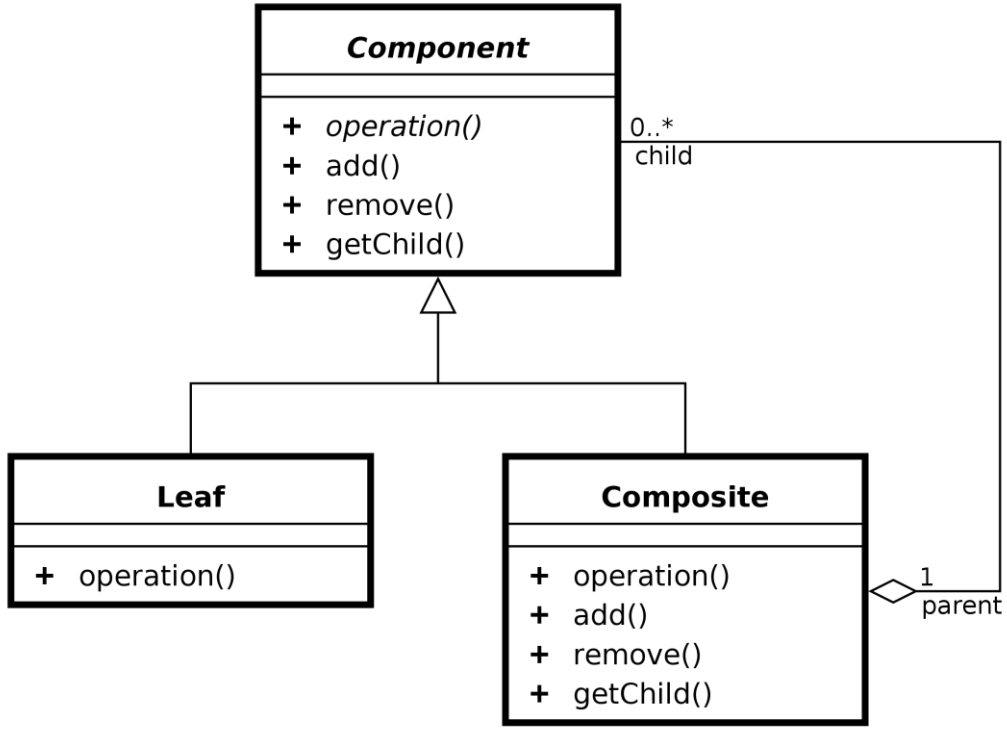
סימון הרשאות של שדות של מחלקה:

הרשאת גישה	**סימון**
Public	+
Private	-
Protected	#
Static	myField

©איילת ויץ 2023

דוגמאות לתרשימי UML





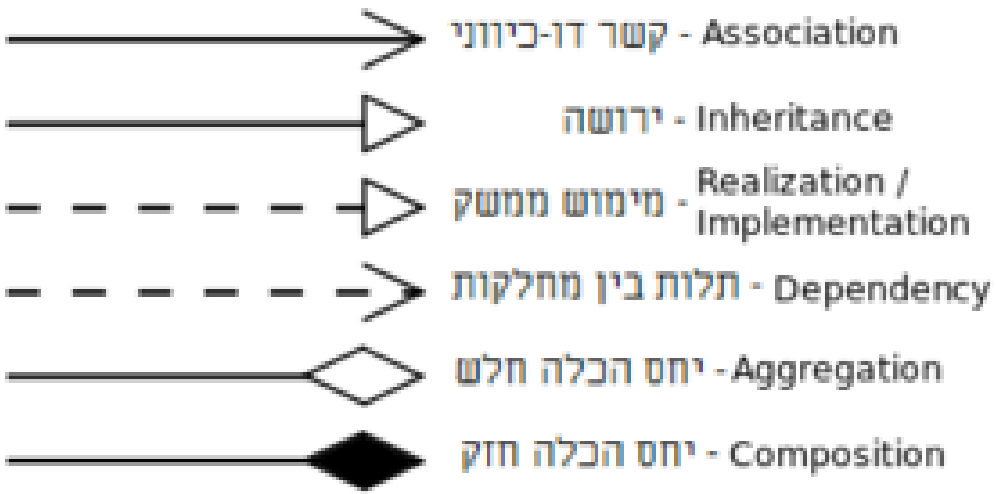
©איילת ויץ 2023

קשרים בין מחלקות בתרשים UML

ייתכנו סוגי קשרים שונים בין מחלקות

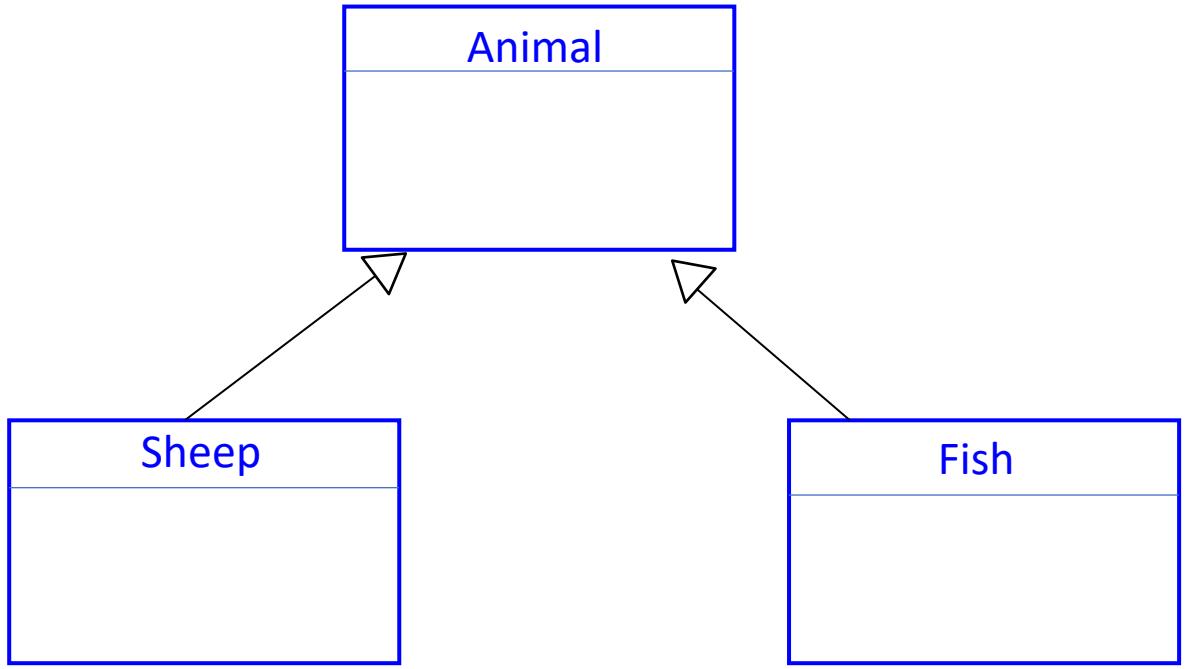
קשר בין מחלקות יסומן בעזרת חץ

לכל סוג קשר יש סימון גרפי שונה בשפת UML



©איילת ויץ 2023

סימון ירושה בתרשים UML



©איילת ויץ 2023

בס"ד

-END-

©איילת ויץ 2023