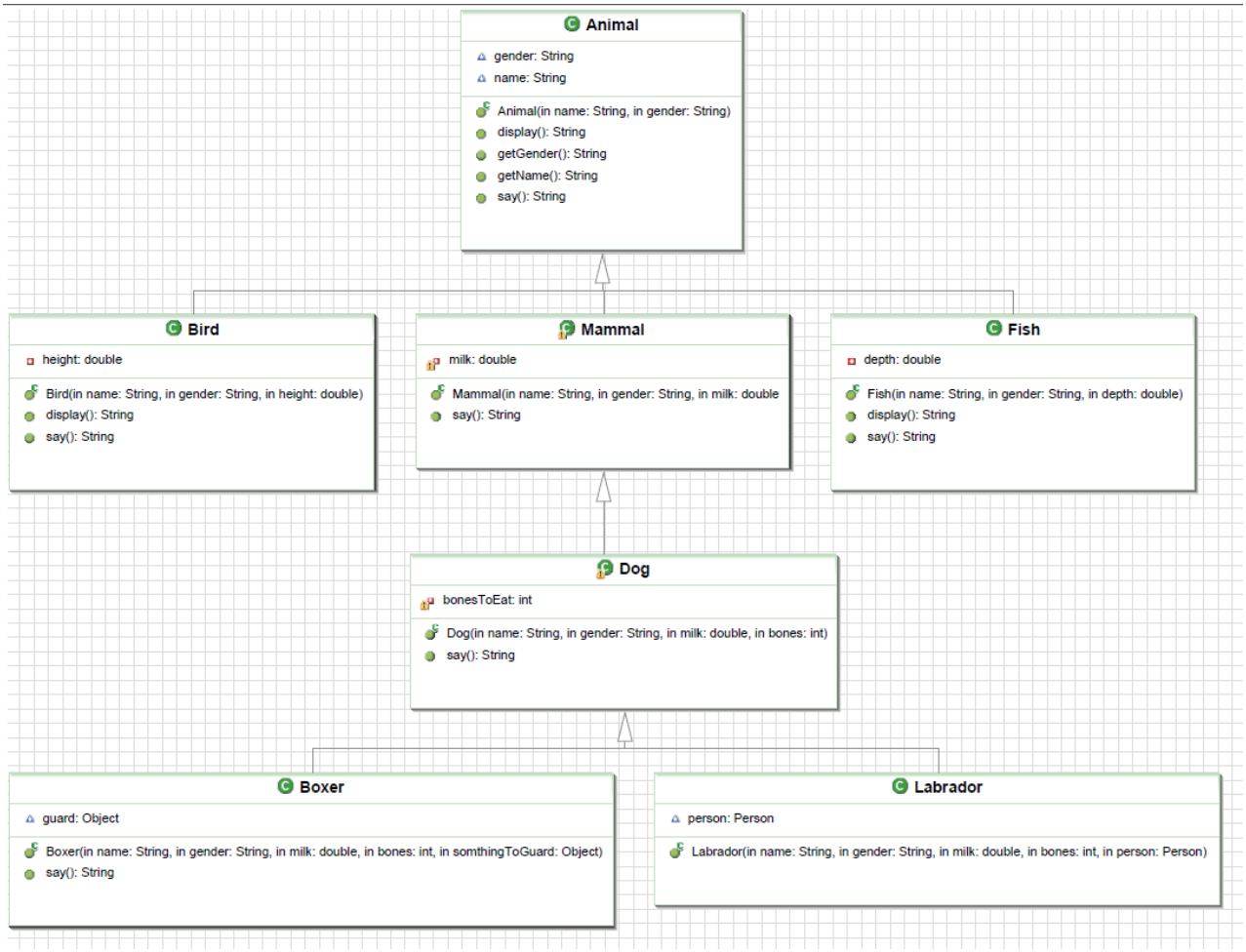
פולימורפיזם

להסתכל על אותו אובייקט בצורות שונות.

אובייקט של תת-מחלקה (המחלקה הנגזרת) הוא גם אובייקט של מחלקת העל (מחלקת הבסיס)
(כלב הוא גם בעל-חיים ושם יונק)



פולימורפיזם מאפשר להשיג שתי מטרות:

- לטפל בצורה <u>אחידה</u> באובייקטים מסוגים שונים כאילו כולם מאותו טיפוס *(טיפול אחיד לכל בעלי החיים מתוך היאחלקות של Animal)*
- התגובה של כל אובייקט לטיפול האחיד תהיה <u>ייחודית</u> לו.

טיפוס הפרמטר לפעולה

אובייקט מתת מחלקה יכול להיות מועבר כפרמטר לפעולה אם הפרמטר של הפעולה הוא מסוג מחלקת העל שלו. למשל: פעולה שהפרמטר שלה הוא Animal יכולה לקבל כל אובייקט מתת המחלקות של Animal כי כולם סוג של בע"ח.

הוגדר האובייקט: `Mammal mam = new Mammal(…);`

הוגדרה הפעולה: `void method (Animal a)`

משפט הזימון הבא יהיה נכון: `method (mam)` מתבצעת **המרה** (אוטומטית) **כלפי מעלה** upcasting

נגדיר מערך zoo של בעלי חיים: Animal [] zoo = **new** Animal [7] ;

ניתן לשים בכל תא במערך אובייקט של המחלקה Animal או אובייקט של אחת המחלקות היורשות מ- Animal.

טיפול אחיד: כל תא יכיל הפנייה לטיפוס נתונים אחר בעץ ההורשה. נוכל להתייחס לכל האובייקטים שנוצרו מתת המחלקות שונות כאילו היו מטיפוס מחלקת העל שלהם *(כי כולם סוג של בעלי-חיים - Animal)*.

גמישות: אם נרצה בעתיד ליצור עוד תת-מחלקות של בעלי-חיים, לא נצטרך לשנות בתכנית, ונוכל להגדיר עצמים נוספים מסוג בעלי-חיים במערך zoo.

תגובה ייחודית: אם לאחת המחלקות יש מימוש שונה לפעולה שמוגדרת ב- Animal, יפעיל העצם את הפעולה שבמחלקה שלו.

```
Animal [] zoo = new Animal [7];

zoo[0] = new Animal   ("animal", "male");
zoo[1] = new Dog      ("Dasty", "female", 0.5, 3);
zoo[2] = new Fish     ("Gupy", "male", 0.3);
zoo[3] = new Bird     ("Twitty", "female", 3.5);
zoo[4] = new Boxer    ("Rexy", "male", 3.2, 3, "house");
zoo[5] = new Labrador ("Toto", "male", 2.5, 4, new Person("Hila"));
zoo[6] = new Mammal   ("Mammal", "female", 2.3);
```

שימו ♥:

- דרך הפנייה מסוג Animal ניתן לגשת רק לתכונות ולפעולות המוגדרות ב- Animal *(ולכן הן משותפות לכל האובייקטים מהמחלקות היורשות)*.
- לא ניתן לגשת לתכונות או פעולות שמוגדרות בתת החלקות *(כי ההפניה מסוג בעלי-חיים לא מכירה אותן)*.
- אי אפשר לבחור טיפוס אחר לאיברי המערך, למשל Dog כי בעלי החיים האחרים אינם סוג של כלב.
- **מי שקובע אלו מהפעולות ניתן להפעיל זה טיפוס ההפניה** *(רק פעולות המוצגות במחלקה של טיפוס ההפניה)*.
  מי שקובע איזו פעולה תופעל זה טיפוס העצם *(אבל - רק פעולות שטיפוס ההפניה מכיר)*.
  כלומר - ניתן לבקש מכל העצמים במערך להפעיל את הפעולה ()say כי היא מוגדרת ב- Animal.
  כל איבר יפעיל את הפעולה הקרובה לו ביותר בעץ ההורשה. *(הפוקס יפעיל את הפעולה שמוצגת בפוקס הלבבות והכלב יפעילו את הפעולה הכתובה בכלב. היונק, הדג, העוף והבעל חיים יפעילו כל אחד את הפעולה המוצגת במחלקה שלו)*.

גיוון פולימורפי של פעולות

תגובה ייחודית: אם לאחת המחלקות יש מימוש שונה לפעולה שמוגדרת ב- Animal, יפעיל העצם של את הפעולה שבמחלקה שלו.
ל- Bird ו- Fish יש פעולות display() משלהם, ולכן הם יפעילו את הפעולה להם.
כל אובייקט שבמחלקה שלו מומשה הפעולה say(), יפעיל את ה- say() שלו.
אם לא קיים מימוש במחלקה שלו, יחפש את ה-say הקרוב אליו ביותר בעץ ההורשה.

```
for (int i = 0 ; i < zoo.Length; i++)
{
    Console.Write(zoo[i].display());
    Console.WriteLine(":\t "+zoo[i].say());
}
Console.WriteLine();
```

כל בע"ח במערך יפעיל את ה- say() וה- display() שלו

הפעולה שתתבצע נקבעת לפי טיפוס האובייקט ולא לפי טיפוס ההפנייה אליו.

```
Dasty   female  :    I am a Dog
Gupy    male    :    I am a Fish
Twitty  female  :    I am a Bied
Rex     male    :    I am a Boxer
Toto    male    :    I am a Dog
Mammal  female  :    I am a Mammal

num of males is: 4
num of Mammal is: 4
```

הצבת אובייקט מסוג תת-מחלקה במערך מסוג מחלקת העל, אינה משנה את האובייקט, אלא רק את הצורה שבה נתייחס אליו.
ניתן להפעיל על האובייקט את כל הפעולות המוגדרות ומוכרות במחלקת-העל (במחלקה של טיפוס ההפניה), אך הוא יגיב בצורה הייחודית לו.

פולימורפיזם מבוסס על הורשה

המרה = שינוי נקודת ההסתכלות על העצם.
ההמרה מתבצעת על ענף בעץ ההורשה *(כלפי מעלה או מטה של ענף ויונק הוא סוג של בעל-חיים, אבל לא סוג של ציפור או דג)*

המרה כלפי מעלה - up casting:

הסתכלות על העצם כאילו היה מטיפוס מחלקת העל. הסתכלות על הכלב ועל היונק כאילו היו בע"ח.
ההמרה נעשית על ידי הפניה מטיפוס מחלקת-העל שאליה ביצענו את ההמרה.

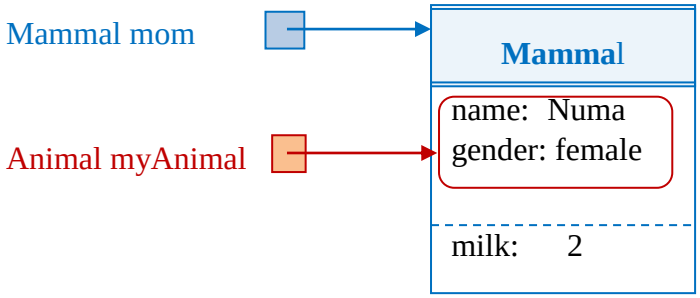
המרה כזו מאפשרת את הטיפול האחיד לאיברים מטיפוסים שונים. (למשל: יצירת מערך של עצמים).

המרה כלפי מעלה נחשבת "המרה בטוחה" כי אובייקט מסוג תת-מחלקה הוא גם סוג של אובייקט מסוג מחלקת העל *(לכל היותר פנינו באמצעות הפנייה מסוג מחלקת-העל לפעולה שאינה מוצהרת במחלקת העל שמקבלת במחלקת הקומפיילציה)*.
המרה כלפי מעלה היא המרה אוטומטית שאינה מחייבת תוספת קוד מיוחדת.

דוגמא:

```
Mammal mom = new Mammal ("Numa", female", 100, 2) ;
Animal myAnimal = mom;
```

ההפנייה myAnimal אינה משנה את העצם mom אלא רק את נקודת ההסתכלות עליו.
באמצעות mom נוכל לבצע את כל הפעולות המוגדרות על יונק.
באמצעות myAnimal נוכל לבצע רק את הפעולות המוגדרות במחלקה בע"ח.



הפעולה: myAnimal.GetMilk() אינה חוקית

כי milk והפעולות עליו מוגדרות רק ב- Mammal ואילו myAnimal הינו הפנייה מסוג Animal.

המרה כלפי מטה - Down Casting:

כיצד נוכל להסתכל על בעלי החיים השונים שבמערך zoo כעל אובייקטים מסוג המחלקה שלהם?

בשתי דרכים:

- **המרה מפורשת:**

ניצור הפנייה מהסוג הרצוי (הפנייה מסוג שבין מחלקת העל והמחלקה של העצם - למשל: עבור Dog נוכל ליצור הפנייה מסוג Animal, Mammal, Dog), נמיר את העצם ונציבו בהפניה:

```
Mammal myMammal = (Mammal) myAnimal;
Labrador myLab = (Labrador) zoo[5];
Dog myDog = (Dog) zoo[5] ;                    וכדי
```

לאחר ביצוע ההמרה, נוכל לפנות לתכונות ולפעולות של העצמים המומרים, לפי סוג ההמרה, כלומר - להתעניין בכמות החלב של myMammal, myLab, myDog ובכמות העצמות של myLab ו- myDog

- **המרה מרומזת לצורך מקומי:**

הסתכלות רגעית לצורך ביצוע הפעולה הנוכחית בלבד:

```
((Mammal)myAnimal).NurseFrom (mom)
```

שימו לב! יש חשיבות לסדר הביצוע ולכן חשיבות לסוגריים סביב העצם וטיפוס ההמרה.
ללא סוגריים אלו תתקבע תחילה הפעולה ואז ינסה להמיר את תוצאת הפעולה לטיפוס יונק, דבר שבוא כמובן לא אפשרי!

ניסיון לבצע המרה על ידי השמה בלבד תגרור הודעת שגיאה:

```
Mammal myMammal = myAnimal ;          //        שגיאה !!!
```

שגיאה מסוג אי התאמה בין טיפוסים: incompatible types.

המרה כלפי מטה נחשבת לא בטוחה כי אובייקט ממחלקת העל אינו בהכרח אובייקט של תת מחלקה.
ההמרה תתבצע רק אם האובייקט המומר אמנם שייך לתת המחלקה אליה מבצעים את ההמרה.
אם אינו שייך לענף שבין מחלקת העל ותת המחלקה, תתקבל **שגיאת זמן ריצה**, התכנית תעצר ותתקבל הודעת השגיאה:
Class Cast Exception - המרה בלתי חוקית.

דוגמא לשגיאה:

```
Animal rob = new Animal ("Rob", "male", 20);
Mammal mamRob = (Mammal) rob ;          //  run time Error
```

rob הוא מטיפוס Animal ולכן לא ניתן להמיר אותו לטיפוס שהוא תת-מחלקה של Animal.
השגיאה תתגלה רק בזמן הריצה ולא בשלב ההידור.
מבחינת המהדר המתכנת ביצע המרה מפורשת. רק בזמן הריצה יתגלה ש- rob אינו מסוג יונק ולכן תיעצר התכנית ותתקבל הודעה שההמרה בלתי חוקית.

האם קטע הקוד הבא תקין?

```
(1)    Mammal goofy = new Mammal ("Goofy", "male", 250);
(2)    Object obj = goofy ;
(3)    Console.WriteLine (obj.GetGender());
(4)    Animal ann = (Animal) obj ;
(5)    Console.WriteLine (ann.GetGender());
(6)    Console.WriteLine ((Bird)obj).GetName());
```

האם ההמרות שבקטע הקוד יגרמו לשגיאות הידור? לשגיאות ריצה? הסבר !!

(2) המרה כלפי מעלה היא המרה בטוחה.

(3) obj אינו מכיר את הפעולות של Animal (או של Mammal) ולכן תתקבל **שגיאת הידור** (שגיאת **קומפילציה**).

(4) ביצענו המרה מפורשת לכן אין שגיאת הידור.

(5) מכיוון שההמרה כלפי מטה שבשורה 4 היא המרה לטיפוס בעל-חיים, ויונק הוא סוג של בעל-חיים, הרי שההוראה תקינה (GetGender() היא פעולה המוגדרת ב- Animal).

(6) **שגיאת זמן ריצה**: obj הוא מטיפוס יונק וכן לא ניתן לבצע עליו המרה לטיפוס עוף.
מבחינה תחבירית, רשמנו קוד תקין ולכן לא תתקבל שגיאת קומפילציה.
השגיאה תתגלה רק בזמן הריצה, כשהמחשב ינסה לבצע א ההמרה.

האופרטור is

אופרטור המאפשר לבדוק בזמן הריצה אם הפנייה מסויימת מפנה לעצם מטיפוס מסוים.

reference **is** Type
משתנה/הפניה **is** טיפוס

הביטוי מחזיר ערך בוליאני.

כדי להימנע מהמרה בלתי חוקית נרשום:

```
if (rob is  Mammal)
{
      Mammal mamRob = (Mammal) rob ;
      mamRob.addMilk(10);
}
```

נשים לב שכל מי שהוא יונק או יורש מיונק עונה להגדרה זו

```
count = 0;
for (int i = 0 ; i < zoo.Length; i++)
    if (zoo[i] is Mammal)
        count ++;
Console.WriteLine("num of Mammal is: "+ count);
```

```
num of Mammal is: 4
```

כיצד נוכל ליצור מערך של טיפוסים בסיסיים מסוג: int, double, float, byte, long ??

לכל אחד מהטיפוסים מחלקה עוטפת המגדירה פעולות בסיסיות שניתן לבצע על טיפוס זה.
מחלקת העל המשותפת היא Object, ולכן ניצור מערך של Object
באותה מידה נוכל להכניס גם ערכים מסוגים טקסטואליים וטיפוסים אחרים למערך זה.

```
Animal a1 = new Boxer    ("Rexy", "male", 3.2, 3, "house");
Animal a2 = new Labrador ("Toto", "male", 2.5, 4, new Person("Hila"));

a1.Say();       // Boxer של Say() ולכן יופעל Say()  הגדיר מחדש את Boxer
a2.Say();       //       Dog של  Say()  ולכן יופעל  Say() אין Labrador -ל

Mammal m1 = (Mammal) a1;                  //       המרה מפורשת
int milk1 = m1.GetMilk();
int milk2 = ( (Mammal)a2).GetMilk()       //       המרה מרומזת

Boxer b1 = (Boxer) a2;          //       Labrador מטיפוס a2   שגיאת הידור!
                                // מתבצעת ההמרה אבל בעת ניסיון ההשמה ל- b1 מתגלה חוסר ההתאמה

Persom owner = ((Labrador) a1).GetPerson();     //       שגיאת זמן ריצה!
                                                //       Boxer מטיפוס a1  ולכן אין לו בעלים
                                                //       השגיאה תתגלה רק בזמן הריצה
```