# Objekt Orienteret Programmering C#

Finn Vilsbæk

fvs@medieskolerne.dk

# Emner for i dag

- Forstå objekter, klasser, attributer og metoder
- Forstå access modifiers
- Skabe en klasse
- Instantiere og referere til objekter
- Lad objekter interagere og udveksle information

- **Læs: *Learning C#, kapitel 6 + 7***

# Klasser

- Vi **klass**ificerer ting I den virkelige verden, for at illustrere hvad tingen er for en størrelse
  - en hund
  - en firkant
  - en person
  - en bruger
- En klasse modellerer en ‘real world entity’
- Klasser har **attributter**
- Klasser har **metoder**

# Klasser

- Instanser af en klasse
  - Fido (en hund), Rufus (en anden hund)
  - 3 x 3 cm (en firkant), 2 x 2 cm (en anden firkant)
  - Thomas (person), John Bo (anden person), Finn (tredje person)
  - Webdev (en bruger), Administrator (en anden bruger)

# Simpleste klasse *evah*

```
class Dog
{
      private string name;
}
```

# Attributter og metode

```
class Dog
{
    private string name;
    public void Bark()
    {
        System.Console.Write(”Woof”);
    }
}
```

# At instantiere en hund

```
class Program
{
      static void main(String args[])
      {
            Dog fido = new Dog();
            fido.Bark();
      }
}
class Dog
{
      private string name;
      public void Bark()
      {
            System.Console.Write(”Woof”);
      }
}
```

# At instantiere flere hunde

```
class Program
{
      static void main(String args[])
      {
      Dog fido = new Dog();
      Dog rufus = new Dog();
      fido.Bark();
      rufus.Bark();
      }
}
class Dog
{
      private string name;
      public void Bark()
      …
}
```

# 3 OOP søjler (pillars)

# 3 OOP søjler

- **Encapsulation:**
  ”Enkapsuler det, som ændrer sig” - ideen er her at holde hver enkelt type / klasse afgrænset og ‘self-contained’, så vi kan ændre implementeringen af en klasse uden at det vil have effect på andre klasser I systemet.

- **Specialization:** ***‘is-a’ relationship***

  Nedarvningstræet i C# tillader specialisering i klasser; for eksempel kan både ‘Kat’ og ‘Hund’ optræde som subklasser til superklassen ‘Pattedyr’ – dette giver os en måde hvorpå vi kan etablere hierarkiske relationer imellem vores klasser.

# 3 OOP søjler

- **Polymorphism:**

  Græsk ord for ‘mange former  / figurer’. Polymorfisme tillader os at behandle en gruppe af  hierarkisk relaterede objekter på same måde, og samtidig at lade disse objekter selv finde ud af at implementere programinstruktionerne.

  Et eksempel:  web controls som en knap og en listbox vil sikkert begge have en ‘Draw()’ metode – men deres individuelle implementation af denne metode vil være forskellig I hvert tilfælde. Når du sætter kontroller op på en webside, behøver du som programmør ikke at vide nøjagtigt hvordan hver subtype klarer at tegne sig selv på websiden -  du behøver blot at vide, at hver type er defineret sådan at man kan kalde en enslydende metode Draw() på begge objekter.

# ...og nu

**...CODE-A-DOG**

**...CODE-A-HUMAN**

**...FÅ KLASSERNE TIL AT INTERAGERE:**
**SEND INFORMATION FEM OG TILBAGE**

# Konstruktøren: Constructor

- Kode der afvikles, når et objekt skabes (instantieres)
- Navnet på konstruktøren skal være det samme som klassens navn
- Konstruktører har ingen return type (ikke engang void)
- Sommetider ser man, at konstruktøren I tutorials og programdokumentation refereres til som ‘CTOR’

# ‘Access modifiers’ - adgang

- Public : synlig for enhver metode  i enhver klasse.
- Private : members i klasse A som får access modifier ‘private’ kan kun ses af metoder internt i klasse A.
- Protected : members i klasse A som får access modifier ‘protected’ er tilgængelige for metoder i klasse A, og også for metoder I klasser, der *nedarver* fra klasse A.

# Planet of the Shapes

Polymorphism - Polymorfisme

Inheritance - Nedarvning

Overridden methods – Overstyrede metoder

# Planet of the Shapes

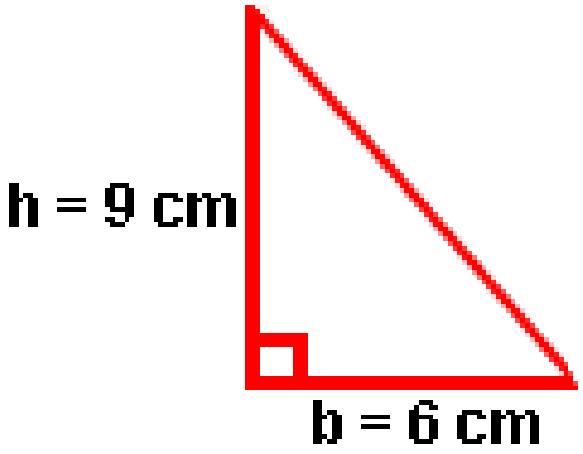
Find the area of a right triangle

$$A = \frac{1}{2} \cdot b \cdot h$$

$$A = \frac{1}{2} \cdot (6 \text{ cm}) \cdot (9 \text{ cm})$$

$$A = \frac{1}{2} \cdot (54 \text{ cm}^2)$$

$$A = 27 \text{ cm}^2$$



# Monstahs, Inc.

# Deep sea monster..

# Uncle Vlad!

# Monstahs, Inc.

Nasty..

Horrible..

Ugly..

>> Lad os få monstrene lidt under kontrol her!

# Monstahs, Inc.

Ændringer til Monster.cs: (tag 'description' ud)

```
public class Monster
{
    public enum DamageLevel
    {
        healthy,
        groggy,
        battered,
        stumbling,
        wasted
    };

    // public properties / members we will want to get and set from the main method
    public DamageLevel currentDamageLevel { get; set; }
    public string name;

    public string partingWords;
    public string victoryWords;

    // it's now a private business proposition to get killed!
    private bool dead { get; set; }
```

# Monstahs, Inc.

Noter:

Ændringerne i Monster.cs på forrige slide introducerede en enum kaldet DamageLevel, som vi kan bruge til at gøre tekst outputtet mere varieret i spillet.

Vi fjerner også 'description' member, og erstatter denne member med to nye strings, som kan bruges til at give et monster dets egen stemme i spillet.

Lav ændringer til 'fight' foreach loopet som vist på de næste slides, og prøv derefter at implementere nogle af de ekstra code challenges der er nævnt i kodekommentarerne.

# I program.cs (del 1):

```
// New version of the fight engine, using a damagelevel enum and with randomized attacks.
foreach (Monster protagonist in myMonsters)
{
    // delay is necessary to stop the random seed from being the same
    System.Threading.Thread.Sleep(10);

    if (protagonist.isDead() == true)
    {
        // do nothing, e's already dead!
        Console.WriteLine("Since " + protagonist.name +
                          " is currently dead, this dude can't attack anybody..");
    }
    else
    {
        Random random = new Random();
        int targetDude = random.Next(4);
        int damage = random.Next(5);
        Console.WriteLine("Kiii-ai! " + protagonist.name +
                          " is now in attack mode, and the damage int is set to: "
                          + damage);

        if (myMonsters[targetDude].isDead() == true)
        {
            // do nothing
            Console.WriteLine("Since " + myMonsters[targetDude].name +
                              " is already dead, this dude can't be killed twice..");
        }
```

# I program.cs (del 2):

```
            // Challenge: can you see a way to do a check here, so a monster can not attack itself?
            // For example, you could check on the name property and see if the names are identical for the
            // attacker and the target monster.

            else
            {
                switch (damage)
                {
                    case 0:
                        myMonsters[targetDude].currentDamageLevel = Monster.DamageLevel.healthy;
                        break;
                    case 1:
                        myMonsters[targetDude].currentDamageLevel = Monster.DamageLevel.groggy;
                        break;
                    case 2:
                        myMonsters[targetDude].currentDamageLevel = Monster.DamageLevel.battered;
                        break;
                    case 3:
                        myMonsters[targetDude].currentDamageLevel = Monster.DamageLevel.stumbling;
                        break;
                    case 4:
                        myMonsters[targetDude].currentDamageLevel = Monster.DamageLevel.wasted;
                        // finish him!
                        myMonsters[targetDude].setDeadMemberTrue();
                        break;

                    default:
                        break;
                }

                Console.WriteLine("Attack: a damage level of " + myMonsters[targetDude].currentDamageLevel +
                                  " is inflicted on " + myMonsters[targetDude].name);
            }
        }
    }
```

# Monstahs, Inc.

Sort challenge - 1:

Lad os prøve at se, om vi kan få monster listen til at sortere sig selv når løkken er afsluttet, så vi kan lade currentDamageLevel bestemme hvem, der er i toppen og bunden af listen. Dette vil lade os bestemme en vinder, ud fra det enkelte monsters aktuelle helbred.

Hint: se http://stackoverflow.com/questions/230588/problem-sorting-lists-using-delegates

# Monstahs, Inc.

Sort challenge - 2:

Når du med succes har sorteret monster listen ud fra currentDamageLevel, så print listen til konsollen:

```
foreach (Monster badguy in myMonsters)
{
    Console.WriteLine(badguy.name +
                      " has a damage level of: "
                      + badguy.currentDamageLevel);
}
```

# Monstahs, Inc.

Weapons challenge:

- Ethvert nogenlunde respektabelt monster vil selvfølgelig gøre brug af et(hvert) våben, hvis der er et sådant til rådighed. Lav en mindre liste af våben, der gives til hvert monster objekt når det instantieres - og når monsteret angriber, så vælg et tilfældigt / random våben fra listen til at angribe med. Vis det valgte våben i text outputtet, så man kan se at det er forskellige våben der vælges.