Campus Nyköping

PROGRAMMERING I .NET C#1 - 10

OOP 2 –Properties, Constructor och Metoder

Förra gången

- Demo av övningar
- Objektorienterad programmering

Idag

- Upprop
- Mer om objektorienterad programmering
  - Klasser
  - Attribut
  - Properties
  - Metoder
  - Konstruktorn

Klasser

- Klasser är modeler av object I den “riktiga” världen och beskriver:
  - Attribut/egenskaper
  - Beteenden (metoder)
- Klassen beskriver objektets struktur
  - Den enskilda objektet beskriver en specific **instans** av en klass
- Egenskaper innehåller relevant information om objektet
- Metoder implementerar objektets beteende

Klasser

- Klasser kan ha medlemmar
  - Fält/attribute, metoder, properties, konstruktorer med mera…
- Medlemmar kan ha åtkomstmodifierare
  - Public, Private, Internal m m
- Medlemmar kan vara static(gemensamma) eller specifika för ett givet objekt (inte static).

Attribut

```
internal class Person
{
    public string name;  // field - Fält
}
```

- Detta gör att alla fält kan läsas av alla metoder, och att man var som helst kan ändra värdet.
- Inkapsling
  - Betydelsen av inkapsling är att se till att "känsliga" data är dolda för användare. För att uppnå detta måste du:
  - deklarera fält/variabler som privata
  - tillhandahålla publika get and set-metoder, via egenskaper, för att komma åt och uppdatera värdet för ett privat fält

Fält och Egenskap

```
internal class Person
{
  private string name;  // field – Fält, lowerCamelCase
  public string Name    // property – Egenskap, UpperCamelCase
  {
     get { return name; } // returnerar värdet på fältet
     set { name = value; } // ger fältet ett värde
  }
}
```

Används såhär:

- `Person person = new Person();`
- `person.Name = "Micke";`
- `Console.WriteLine(person.Name);`

Constructor

- En konstruktor är en speciell metod som används för att initiera objekt.
- Fördelen med en konstruktor är att den körs när ett objekt i en klass skapas.
- Den kan användas för att ställa in initialvärden för fält.
- Namnet på constructor-metoden är samma som klassen.
  - Om Klassen heter Person, så heter även konstruktorn Person.
- Det finns en "osynlig" och tom constructor när man skapar en klass.

```
internal class Person
{
        private string name;
        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        public Person() // Tom constructor
        {
        }
}
```

Använda en constructor

```
internal class Person
{
    private string name;  // Field - Fält
    public string Name    // Property - Egenskap
    {
        get { return name; } // returnerar värdet på fältet
        set { name = value; } // ger fältet ett värde
    }
    public Person(string name) // Constructor
    {
        this.name = name;
    }
}
```

- Detta anropas sedan med:

```
Person person = new Person("Micke"); // set
Console.WriteLine(person.Name); // get
```

Metoder i ett objekt

- Det går, som vi tidigare sett, att placera en metod I en klass, men om vi gör ett object av en klass, hur ser det ut då?

```
 public string SayHello() // Observera, ingen static
 {
     string helloText = "Hejsan " + this.name + ", hoppas du mår bra.";
     return helloText;
 }
Används så här:
 string helloText = person.SayHello(); // Ingen inparameter, objektet vet
redan name
 Console.WriteLine(helloText);
```

Hela Person-exemplet

- Program.cs

```
Person person = new Person("Micke");
Console.WriteLine(person.Name);
string helloText = person.SayHello();
Console.WriteLine(helloText);
```

- Person.cs

```
internal class Person
{
        private string name;
        public string Name;
        {
          get { return name; }
          set { name = value; }
        }
        public Person(string name)
        {
            this.name = name;
        }
        public string SayHello()
        {
            string helloText = "Hejsan " + this.name;
            return helloText;
        }
}
```

Egenskap – Property - Kortversion

- Istället för allt detta:

```
private string name;  // field – Fält
public string Name    // property – Egenskap
{
      get { return name; } // returnerar värdet på fältet
      set { name = value; } // ger fältet ett värde
}
```

- Så skriver vi bara:

```
public string Name { get; set; }
```

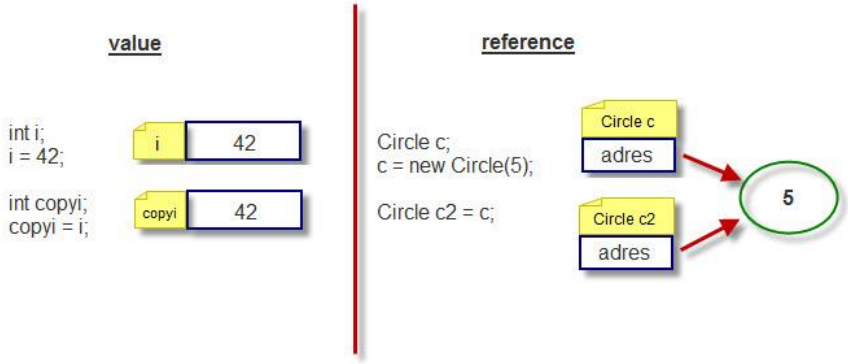
(I Visual studio finns ett kortkommando: prop <TAB><TAB>)

Code along – Properties, constructor och metoder I klasser

- Kod: PropertyConstructorDemo

Referens vs värde-datatyper

- Det finns två typer av datatyper
  - Value
    - En datatyp är en **värdetyp** om den innehåller ett datavärde i sitt eget minnesutrymme. Det betyder att variablerna för dessa datatyper direkt innehåller värden.
    - När du skickar en värdetypsvariabel från en metod till en annan skapar systemet en separat kopia av en variabel i en annan metod. Om värdet ändrades i en metod skulle det inte påverka variabeln i en annan metod.
    - Exempel: bool, decimal, float, int
  - Reference
    - Till skillnad från värdetyper lagrar en **referenstyp** inte sitt värde direkt. I stället lagras adressen där värdet lagras. Med andra ord innehåller en referenstyp en pekare till en annan minnesplats som innehåller data.
    - När du skickar en referenstypsvariabel från en metod till en annan skapas ingen ny kopia, I stället skickar den variabelns adress. Så, om vi ändrar värdet på en variabel i en metod, kommer det också att återspeglas i anropsmetoden.
    - Exempel: Arrays (även om deras element är värdetyper), **Class**, Delegate, (String).



Reference variable

När du skickar en referenstypvariabel från en metod till en annan skapas ingen ny kopia. I stället passerar den variabelns adress. Så om vi ändrar värdet för en variabel i en metod kommer det också att återspeglas i anropande metod.

```
static void EnAnnanMetod(Student std2)
{
    std2.StudentName = "Steve";
}
static void Main(string[] args)
{
    Student std1 = new Student();
    std1.StudentName = "Bill";
    EnAnnanMetod(std1);
    Console.WriteLine(std1.StudentName);
}
```

- ”Steve” kommer att skrivas ut

CODE ALONG- THE ROAD

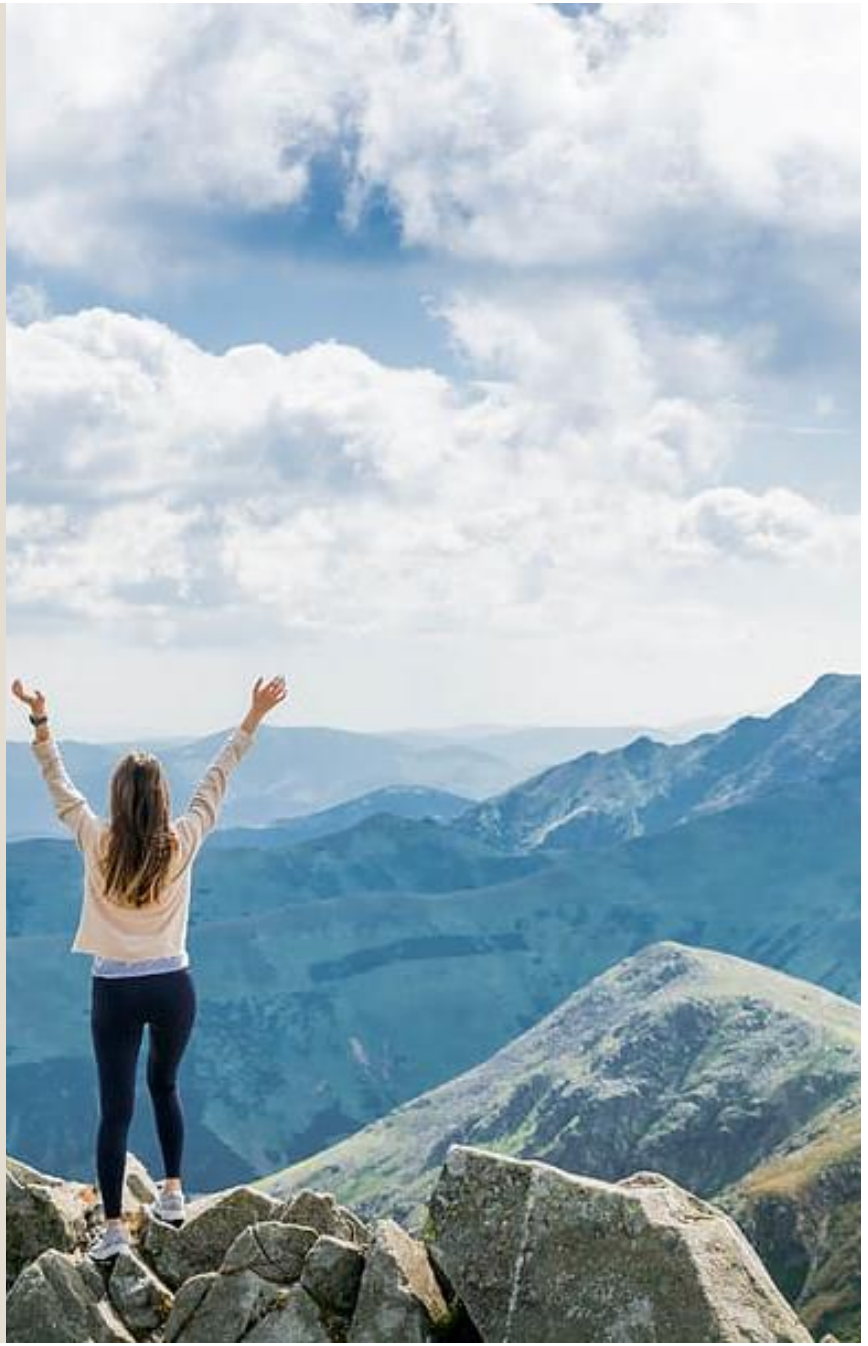
Kod: TheRoad.zip

Övning – Filmlistan del 2

- Gör övningen Filmlistan, men använd I stället properties enligt exempel:
  - Public string Name {get; set;} osv…
- Bygg en constructor, så det är lätt att skapa nya filmer.
- Skapa en metod i Klassen, som skapar strängen som beskriver filmerna.

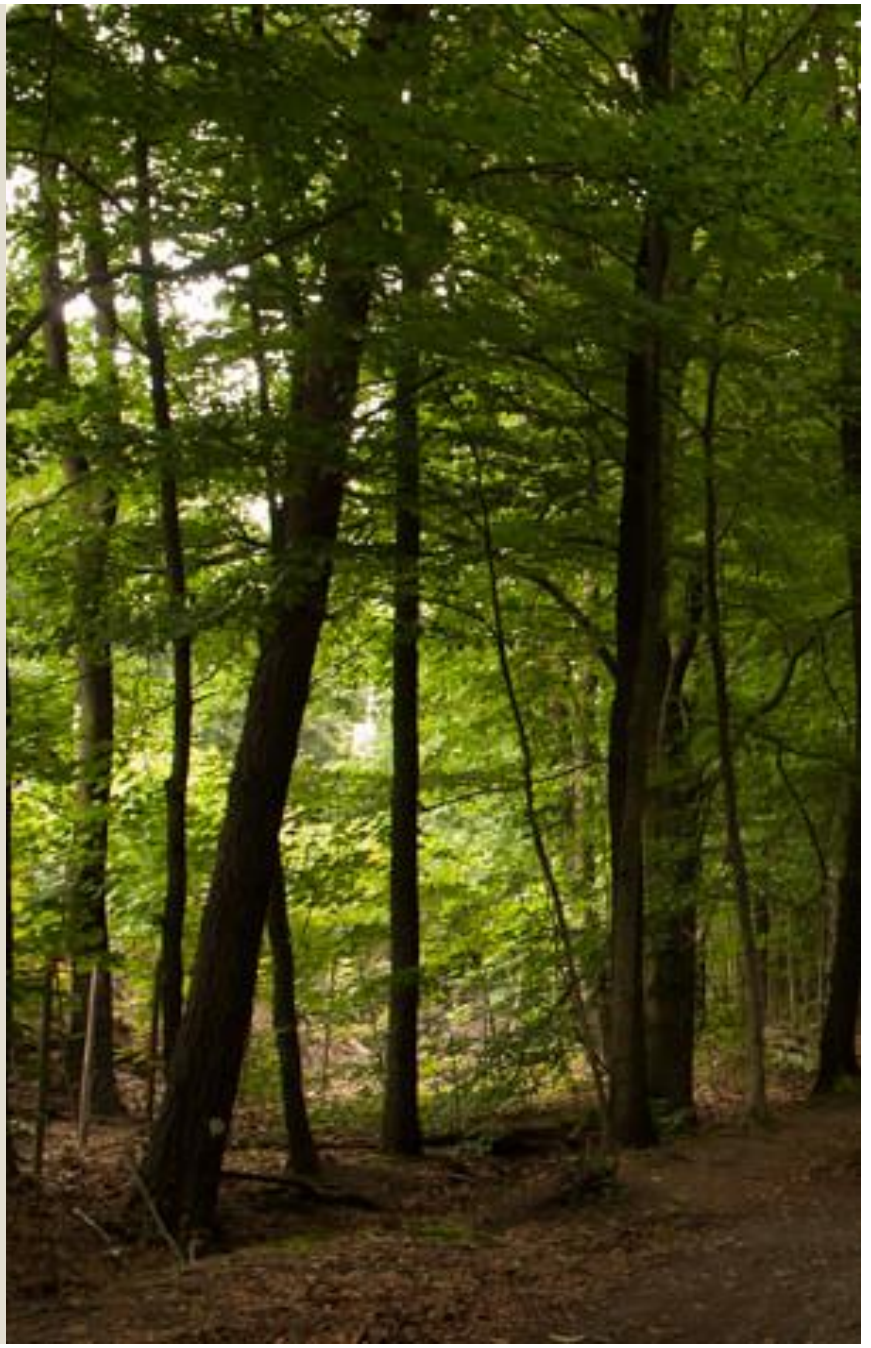
Övning – Herre på täppan

- Herre på täppan
  - Skapa fem personer med slumpmässig styrka
  - Låt den första personen gå upp på berget
  - Låt därefter varje person en och en gå upp
    - De som är starkare kan slå ut den person som finns på berget.
    - De som är svagare blir bortjagade
- Exempel:
  - Johan, med styrka 25 går upp på berget
  - Därefter kommer Nisse, med styrka 18 upp, men blir bortjagad.
  - Sen kommer Agnes, med styrka 34, och Johan blir bortjagad
  - Osv…
- Teknik: Använd properties och constructor



Övning - Skogen

- Skapa ett djur-objekt som har följande egenskaper:
  - string Name – Namn ska vara Varg, Fladdermus, Delfin, Örn, Häst och några till om du vill.
  - bool Nocturnal – Nattdjur, sant eller falskt
  - string Movement – hur den rör sig, t ex springer, simmar, flyger, letar
- Skapa minst en instans av varje djur, gärna fler, och stoppa i en List<Animal> som du döper till forrest.
- Välj två tangenter, som ska vara dag och natt.
  - Trycker du på nattknappen ska det stå följande om vargen och fladdermusen (och andra nattdjur):
    - "Vargen smyger omkring och letar sitt byte" och "Fladdermusen flyger runt bland träden I jakt på mat".
    - De djur som inte är nocturnal beskrivs med: Hästen sover, Delfinen sover osv.
  - Trycker du på dagknappen så ska det omvända ske
    - Vargen och Fladdermusen sover, och de andra djuren gör något, t ex letar mat eller springer genom skogen o s v



Övning – Klubb-Epidemin

- ◦ Skapa ett objekt: **Person**, med följande egenskaper
  - Smittad (bool)
  - SmittadNär (int, personen ska bli frisk igen efter 5 timmar.)
  - Immun (bool, en smittad person som blivit frisk kan inte bli sjuk igen)
- Skapa en lista: **club**, där du stoppar in 20 personer.
  - Ange att EN person är sjuk.
  - Loopa igenom listan och anta att varje sjuk person smittar 1(en) annan person per timme.
  - Första timmen smittar alltså den enda sjuka personen en annan person.
  - Andra timmen smittar dessa två personer ytterligare varsin person
  - Tredje timmen smittar alla fyra ytterligare fyra personer…osv
  - När 5 timmar gått blir första personen frisk, och smittar ingen längre (och är immun), men de andra fortsätter smitta…
  - Stega fram timmarna med en tangenttryckning, en timme per tryck.
- Visa I konsollen vilka som är smittade, och hur många som är immuna.
- Visa också hur många timmar som går.
- Hur många timmar tar det innan alla är immuna?

Länkar

- C# Class Members (Fields and Methods) (w3schools.com)
- C# Properties (Get and Set) (w3schools.com)
- C# Constructors (w3schools.com)
- Value Type and Reference Type (tutorialsteacher.com)