Objektorienterad programmering
Föreläsning 7

Mahmud Al Hakim
mahmud@webacademy.se
www.webacademy.se

Agenda

- Konstanter och readonly
- Statiska klasser
- Standardklassen Math
- Parameteröverföring
- Referensen This
- Tilläggsmetoder
- Uppräkningstyper

Konstanter

- Konstanta uttryck deklareras med nyckelordet **const** t.ex.

  const string Rubrik = "FAKTURA";
  Rubrik kommer alltid att innehålla samma värde!
- När man i en klass eller struct deklarerar en variabel som konstant med hjälp av ordet const kommer denna variabel att automatiskt bli en statisk variabel.
- En variabel som är deklarerad som **const måste alltid initieras direkt i deklarationen.**
- Konstanter ska helst börja med en stor bokstav.

Readonly

- En variabel som är deklarerad som **readonly** får, liksom const-variabler, inte ändras!
- readonly-variabler får initieras direkt i deklarationen eller i en konstruktor.
- Ex

```
// Instansvariabel
public readonly int kundNr;
// Konstruktor
public Konto (int kund) {
   kundNr = kund;
}
```

Statiska klasser

- Statiska klasser innehåller bara statiska metoder.
- Statiska klasser används endast för att kapsla in statiska metoder.
- OBS! Inga objekt får skapas av en statisk klass.
- För att markera att inga objekt får skapas av en klass skriver man ordet static först i klassdeklarationen.
- Ex.

```
static StatiskKlass
{
  …
}
```


Statiska klasser - Exempel

```
static class Funktioner
{
    // Medel är en statisk metod som returnerar medelvärdet av två tal
    public static double Medel(double x, double y)
    {
        return (x + y) / 2;
    }
    // KvadSum är en statisk metod som returnerar summan av argumentens kvadrater
    public static double KvadSum(double x, double y)
    {
        return x*x + y*y;
    }
}
```

Statiska klasser – Exempel fort.

```
class FunktionerTest
{
    static void Main(string[] args)
    {
        double x = 2;
        double y = 7;
        Console.Write("Medelvärdet av " + x + " och " + y + " är: ");
        Console.WriteLine(Funktioner.Medel(x, y));
        Console.Write("Kvadratsumman av " + x + " och " + y + " är: ");
        Console.WriteLine(Funktioner.KvadSum(x, y));
        Console.ReadKey();
    }
}
```


Standardklassen Math

- Math är en mycket användbar **statisk klass** som finns i namnrymden System.
- I denna klass finns ett antal statiska metoder för att beräkna vanliga matematiska funktioner t.ex. kvadratrötter, exponenter och logaritmer.
- Med hjälp av metoderna Min och Max kan man beräkna minsta respektive största talet av två tal.
- OBS! För att använda metoderna måste skriva klassnamnet först t.ex.
  Math.min(a,b);

Standardklassen Math - Exempel

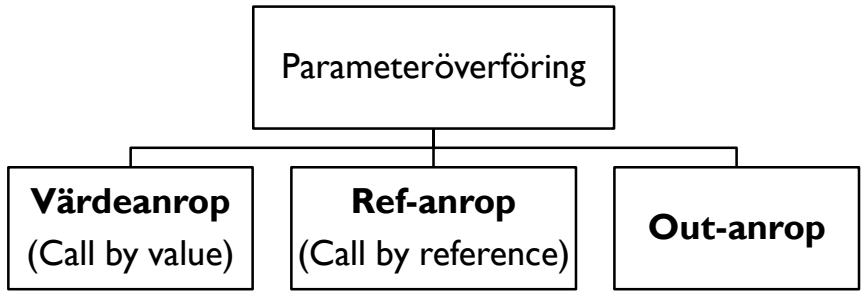
```
class Program
{
    static void Main(string[] args)
    {
        double pi = Math.PI; // Ger talet Pi = 3.14159...
        Console.WriteLine("Pi = " + pi);

        // Metoden Round() avrundar till heltal
        // eller ett antal bestämda decimaler
        Console.WriteLine("Pi = " + Math.Round(pi, 4));

        // Metoden Sqrt(x) ger Kvadratroten ur ett tal
        Console.WriteLine("Roten ur 100 = " + Math.Sqrt(100));

        Console.ReadKey();
    }
}
```


Parameteröverföring



Värdeanrop (call by value)

- Standardmekanismen för parameteröverföring är s.k. värdeanrop (defalut i C#).
- Parametrarna blir **kopior** av de argument som ges vid anropet.
- Ändringar i parametrarna påverkar inte argumenten.
- OBS! Om ett argument är en referens till ett objekt kopieras referensen och objektet den refererar till kan ändras.

Värdeanrop – Exempel 1

```
class C {
    public static void ÖkaInt(int n){
        n++;
    }
}
class ParameterTest{
    static void Main(){
        int i = 0;
        C.ÖkaInt(i);
        // OBS! Värdeanrop.
        // En kopia av i skickas till metoden
        Console.WriteLine("i = " + i);
        Console.ReadKey();
    }
}
```

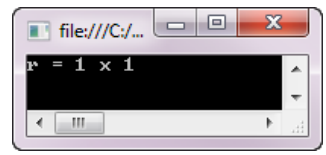
Värdeanrop – Exempel 2

```
struct Punkt{   // OBS! En struct
    public double x;
    public double y;
}
class C {
    public static void ÖkaPunkt(Punkt p){
        p.x++;
        p.y++;
    }
}
class ParameterTest{
    static void Main(){
        Punkt p = new Punkt();
        C.ÖkaPunkt(p);
        // OBS! Värdeanrop.
        // En kopia av p skickas till metoden
        Console.WriteLine("p = " + p.x + "," + p.y);
        Console.ReadKey();
    }
}
```


Värdeanrop – Exempel 3 (Viktigt)

```
class Rektangel{  // OBS! En klass
    public double bredd;
    public double höjd;
}
class C {
    public static void ÖkaRektangel(Rektangel r)
    {
        r.bredd++;
        r.höjd++;
    }

}
class ParameterTest{
    static void Main(){
        Rektangel r = new Rektangel();
        C.ÖkaRektangel(r);
        // OBS! Värdeanrop.
        // En kopia av referensen r skickas till metoden
        Console.WriteLine("r = " + r.bredd + " x " + r.höjd);
        Console.ReadKey();
    }
}
```



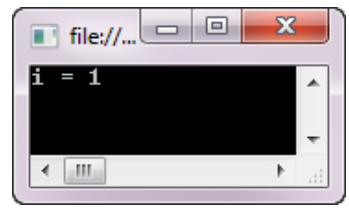
Referensanrop (call by reference)

- Om man verkligen vill att argument av värdetyp skall kunna ändras så kan man använda **ref-parametrar.**
- Ordet **ref** skrivs både i parameterlistan och i anropen.
- Parametrarna blir **referenser till** de argument som ges vid anropet.
- Argumenten måste vara ändringsbara variabler.
- OBS! **Argumenten måste tilldelas något värde**.

Referensanrop – Exempel

```
class C {
    public static void ÖkaInt(ref int x){
        x++;
    }

}
class ParameterTest{
    static void Main(){
        int i = 0;
        C.ÖkaInt(ref i);
        // OBS! Referensanrop
        // En referens till i skickas till metoden
        Console.WriteLine("i = " + i);
        Console.ReadKey();
    }
}
```



Out-anrop

- Samma som ref-anrop men **argumenten behöver inte tilldelas något värde**.
- Parametrarna blir **referenser till** de argument som ges vid anropet.
- Ordet **out** skrivs både i parameterlistan och i anropen.
- Argumenten måste vara ändringsbara variabler.
- OBS! **Parametrarna måste ges värden i metoden**.

Out-anrop – Exempel

```
struct Punkt{   // OBS! En struct
    public double x;
    public double y;
}

class C {
    public static void LäsPunkt(out Punkt p){
        Console.Write("Ange x: ");
        p.x = double.Parse(Console.ReadLine());
        Console.Write("Ange y: ");
        p.y = double.Parse(Console.ReadLine());
        // OBS! Parametrarna måste ges värden i metoden
        // I annat fall visas ett felmeddelande
    }

}
```

Out-anrop – Exempel fort.

```
class ParameterTest{
    static void Main(){
        Punkt p; // Behöver inte tilldelas något värde.
        C.LäsPunkt(out p);
        Console.WriteLine("p = " + p.x + " , " + p.y);
        Console.ReadKey();
    }
}
```


Referensen this

- Alla instansmetoder får en extra dold parameter som inte syns!
- Denna parameter heter **this**.
- Typen på den dolda parametern this är ”referens till en klass eller struct”.
- Referensen this initieras automatiskt så att den vid varje anrop kommer att referera till det aktuella objektet.
- OBS! Statiska metoder och egenskaper har inte någon referens this. Dessa är ju inte knutna till något speciellt objekt.

this - Exempel

```
struct Cirkel
{
    public double x, y, radie;

    public Cirkel(int x, int y, int radie)
    {
        this.x = x;
        this.y = y;
        this.radie = radie;
    }
}
```


Tilläggsmetoder

- Ibland händer det att man tycker att någon instansmetod saknas i en standardklass eller en klass som man inte skrivit själv och som redan används av andra.
- Man kan då skriva en tilläggsmetod (extension method).
- Tilläggsmetoder måste vara statiska och deklareras i en statisk klass.
- Ordet **this** framför den första parametern anger att man vill skapa en tilläggsmetod till den typ som den första parametern har.

Tilläggsmetoder - Exempel

```
static class StringExtra
{
    // Print är en tilläggsmetod till string
    public static void Print(this string s)
    {
        Console.WriteLine(s);
    }
}
class Program
{
    static void Main(string[] args)
    {
        string text = "Ole dole doff";
        text.Print();

        Console.ReadKey();
    }
}
```


Uppräkningstyper (enum)

- Uppräkningstyper deklareras med ordet **enum**.
  enum Betyg { F , E , D , C , B , A }
- Uppräkningstyper representeras med en underliggande heltalstyp.
- Normalt används heltalen 0,1,2 osv. men man kan själv ange heltalsvärden t.ex.
  enum RomerskSiffra
  { I=1, V=5, X=10, L=50, C=100, D=500, M=1000 }
- Uppräkningstyper kan tilldelas och jämföras.