Objektorienterad programmering
Föreläsning 8

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

Agenda (halvdag)

- Objektorienterad programutveckling
  - Algoritmer
    - Algoritmkonstruktionerna
  - Relationer
    - Har-relation (komposition)
    - Känner-till-relation
    - Är-relation (arv)
  - Klassen Object

Algoritmer

- En **algoritm** är en beskrivning av hur ett visst problem lösas.
- En algoritm är en **beräkningsmetod**.
- En algoritm består av ett antal elementära operationer och anvisningar om i vilken ordning operationerna skall utföras.
- Man kan ställa krav på en algoritm:
  - Den skall lösa det givna problemet.
  - Den skall vara entydig (inte luddigt formulerad).
  - Om problemet har ett slutmål så skall algoritmen avslutas efter ett ändligt antal steg.

De viktigaste Algoritmkonstruktionerna

- **Sekvens**
  En sekvens är en följd av steg som utförs ett i taget i den ordningen de skrivits. Varje steg utförs exakt en gång.
- **Selektion**
  En selektion innebär att ett alternativ bland två eller fler skall väljas.
- **Iteration**
  En del av algoritmen skall kunna upprepas ett visst antal gånger eller tills ett visst villkor har uppfyllts.

Stegvis förfining

- När man skall lösa ett komplicerat problem är det till stor hjälp att dela in problemet i mindre delproblem som löses var för sig.
- Delproblemen kan i sin tur behöva delas upp i ytterligare delproblem osv.
- Detta är en mycket viktig teknik vid algoritm- och programkonstruktion!
- Tekniken kalls **stegvis förfining**.
- Man brukar ofta använda den engelska termen **top down design.**
- Tips! Bra exempel finns i boken, sid. 162

Algoritmer - Exempel

- **Beskriv en algoritm som visar hur man kan beräkna summan 1+2+3…+n när n är givet. n är ett heltal > 0**

1. Sätt summa till 0 och en räknare till 1
2. Upprepa följande tills k är större än eller lika med n
   2.1 Addera summa och k och spara resultatet i summa
   2.2 Öka k med ett
3. Det önskade resultatet finns nu i summa

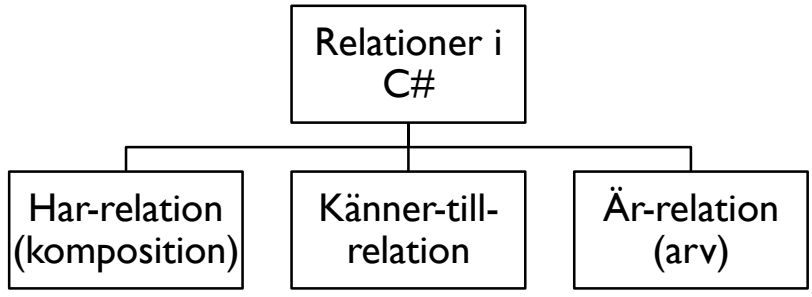
Övning

Lös problemet i C# med hjälp av ovanstående algoritm.

```
class Program
{
    static double Summa(int n)
    {
        double summa = 0;
        int k = 1;
        while (k <= n)
        {
            summa += k;
            k++;
        }
        return summa;
    }

    static void Main(string[] args)
    {
        Console.WriteLine( Summa(5) );
        Console.ReadKey();
    }
}
```


Relationer i C#



Har-relation (komposition)

- När man bygger modeller av verkligheten finner man ofta att vissa objekt är uppbyggda med hjälp av andra objekt.
- Det som kännetecknar en har-relation är att ett objekt har ett annat objekt som en del.
- För att få en äkta har-relation måste instansvariabeln vara av värdetyp (enkel variabel eller struct).
- Exempel
  En rät linje i ett koordinatsystem som har två punkter.


```
// Punkt är en struct som beskriver en punkt
// i ett vanligt tvådimensionellt koordinatsystem
struct Punkt
{
    public double x;
    public double y;
    public override string ToString()
    {
        return "(" + x + ";" + y + ")";
    }
}

// Linje är en klass som beskriver en rät linje
// En linje har en startpunkt och en slutpunkt
class Linje
{
    public Punkt p1, p2;
    public override string ToString()
    {
        return p1 + " " +  p2;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Linje l = new Linje();
        l.p1.x = 1;
        l.p1.y = 2;
        l.p2.x = 3;
        l.p2.y = 4;
        Console.WriteLine(l);
        Console.ReadKey();
    }
}
```


Känner-till-relation

- Känner-till-relationer skapas i C# med hjälp av referenser (jämför med har-relationer).
- Exempel
  Klassen Person. En person har ett namn och en adress. Om vi lägger till en relation som säger att en person kan vara gift, med hjälp av en referens till en annan person, så kommer vi att få en känner-till-relation.

```
class Person
{
    string namn, adress;
    Person makeMaka; // referens till en annan person

    public Person (string namn) // Konstuktor
    {   this.namn = namn;   }

    // Egenskaper
    public string Namn {
        get {   return namn;      }
    }
    public string Adress {
        get {   return adress;    }
        set {   adress = value;   }
    }

    // Metoder
    public void Bröllop (Person p){
        makeMaka = p;       // ordna en referens till maken
        p.makeMaka = this;  // låt maken referera till denna person
    }
    public void Skilsmässa(){
        makeMaka.makeMaka = null;
        makeMaka = null;
    }
    public Person GiftMed(){
        return makeMaka;
    }
}
```



```
class Program
{
    static void Main(string[] args)
    {
        Person p1 = new Person("Anders");
        Person p2 = new Person("Lotta");
        p1.Bröllop(p2);
        Console.Write(p1.Namn);
        Console.Write(" är nu gift med ");
        Console.WriteLine(p1.GiftMed().Namn);
        Console.ReadKey();
    }
}
```


Är-relation (arv)

- När man deklarerar en ny klass kan man utgå från en redan existerande klass och lägga till egenskaper eller förändra egenskaper.
- Man säger då att den nya klassen blir en subklass till den gamla och att den gamla blir en superklass till den nya..
- I C# används termen **derived class** istället för subklass och termen **base class** istället för superklass.
- En av huvudidéerna med arv är att man vid objektorienterad programmering skall kunna använda sig av och ärva färdiga klasser som ingår i s.k. klassbibliotek.
- Exempel: En lärare är en person.


```
class Person {
  protected string namn, adress;   // Protected = synliga i subklassen
  protected int ålder;             // Obs! protected

  protected Person() {} // måste finnas för subklassens skull

  public Person(string namn, string adress, int ålder) {
    this.namn = namn;
    this.adress = adress;
    this.ålder = ålder;
  }

  public string Namn {
    get { return namn; }
  }

  public string Adress {
    get { return adress; }
    set {adress = value;}
  }

  public int  Ålder {
    get { return ålder; }
  }
}
```

```
class Lärare : Person {
    int antalKurser; // ny instansvariabel

    public Lärare(string namn, string adress, int ålder) {
        this.namn = namn;
        this.adress = adress;
        this.ålder = ålder;
    }
    public int AntalKurser {
        get { return antalKurser; }
        set { antalKurser = value; }
    }
}
```

```
class Program
{
    static void Main(string[] args)  {
        Lärare l1 = new Lärare("Mahmud", "", 41);
        l1.AntalKurser = 2;
        Console.WriteLine(l1.Namn+" har " + l1.AntalKurser + " kurser");
        Console.ReadKey();
    }
}
```


Klassen Object

- Klassen Object är automatiskt superklass till alla andra klasser, både standardklasser och sådana man deklarerar själv.
- Klassen Object är även superklass till alla värdetyper t.ex. structer.
- I klassen Object finns metoder som ärvs av alla klasser och structer, bl. a. **ToString** och **Equals**.
- När man anropar ToString för ett visst objekt får man som resultat en text som innehåller namnet på den klass eller struct objektet tillhör.
- Man kan deklarera om metoden ToString i sin egen klass eller struct.

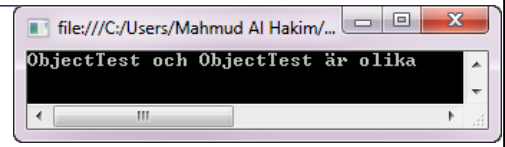
Metoden Equals

- Metoden Equals jämför det aktuella objektet med ett annat objekt.
- Den ger värdet true om den uppfattar objekten som lika och värdet false annars t.ex.

  If (obj1.Equals(obj2))
- Equals jämför bara referenser vid jämförelse mellan referenstyper (t.ex. klasser). Alltså två objekt betraktas lika bara om de var samma objekt.
- Equals jämför objektens innehåll vid jämförelse mellan värdetyper. Alltså två objekt är lika om de har samma typ och innehåller samma sak!

Jämförelse mellan referenstyper

```
1   using System;
2
3   class ObjectTest
4   {
5   }
6   class Program
7   {
8       static void Main(string[] args)
9       {
10          ObjectTest o1 = new ObjectTest();
11          ObjectTest o2 = new ObjectTest();
12
13          if (o1.Equals(o2) )
14              Console.WriteLine(o1 + " och "+ o2 + " är lika" );
15          else
16              Console.WriteLine(o1 + " och "+ o2 + " är olika" );
17
18          Console.ReadKey();
19          }
20  }
```



Jämförelse mellan värdetyper

```
1   using System;
2
3   struct ObjectTest
4   {
5   }
6   class Program
7   {
8       static void Main(string[] args)
9       {
10          ObjectTest o1 = new ObjectTest();
11          ObjectTest o2 = new ObjectTest();
12
13          if (o1.Equals(o2) )
14              Console.WriteLine(o1 + " och "+ o2 + " är lika" );
15          else
16              Console.WriteLine(o1 + " och "+ o2 + " är olika" );
17
18          Console.ReadKey();
19          }
20  }
```