MALMÖ HÖGSKOLA
Fakulteten för Teknik och samhälle

Programmering med C#, grundkurs

En första introduktion till klass och objekt i C#

Farid Naisan
Universitetsadjunkt
farid.naisan@mah.se

sep 2013

Innehållsförteckning

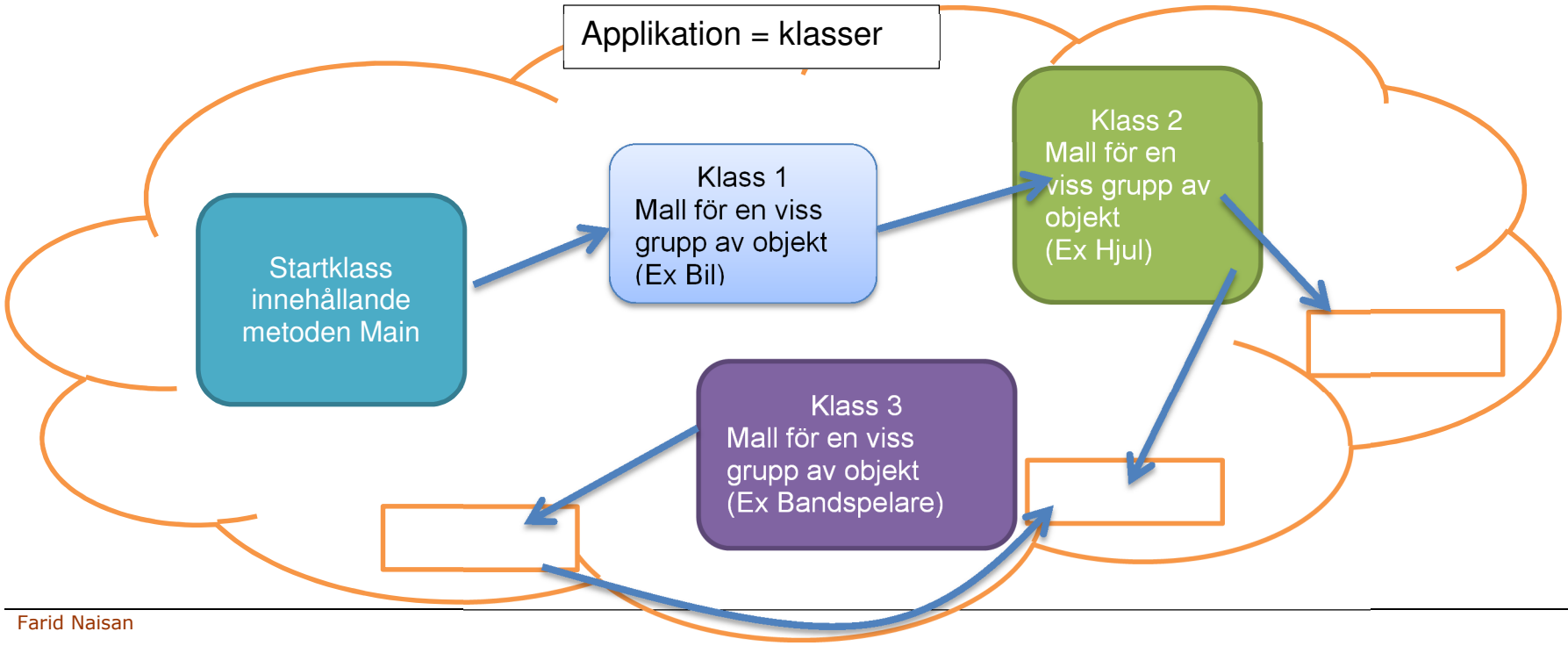
Klass, objekt och applikation 3
1. Applikation 3
2. Klass och objekt 4
3. Delar av en klass 7
4. Sammanfattning 14

Klass, objekt och applikation

1. Applikation

En applikation är en mjukvara, ett program. Från en programmerarens synvinkel är en applikation alla de komponenter, klasser och andra typer som tillsammans bygger ett program. För en användare, är en applikation är det program som den kör och som ofta består av en exe fil, dll:er och andra binära data filer samt textfiler.

En C# applikation består endast av fem typer: **klasser**, **interface**, **delegater**, **struct** och **enum**, huvuddelen av en applikation består av klasser. I denna kurs kommer vi att lära oss att skriva klasser, struct och enum men vi lämnar interface och delegater till fortsättningskursen.



2. Klass och objekt

I OOP-diskussioner talas det mycket om objekt och klasser utan att tänka på om det är klass eller objekt man menar. Betydelsen beror mycket på sammanhanget. En klass är endast en definition av en grupp av objekt. En klass liknar en kakform eller en gjutform som definierar **egenskaper och beteenden** hos en grupp av liknande objekt. Egenskaperna definieras med hjälp av s.k. instansvariabler (eller fält) och beteenden definieras som metoder.

Man kan tänka sig en klass **Bil** som allmänt beskriver en bilar med värden på färg, motorkraft, årsmodell, pris, samt de operationer som förväntas från ett bilobjekt, t.ex. gasa, bromsa, krocka och skrota. Med Bil som klass, kan vi skapa många bilexemplar (instanser) och varje exemplar blir ett unikt föremål (objekt). Min bil och din bil är två unika exemplar av klassen Bil. Dessa har var sin uppsättning av värden på färgen, motorkraften, årsmodellen osv, men de utför samma operationer. Min och din bil är två **instanser** av klassen Bil. Klasser är alltså definitioner för objekt på kompileringsnivå men objekt är unika instanser som skapas i minnet under körning av program.

Det första steget vid OOP-design är att bestämma vilka objekten är och vilka relationer de ska ha. Vad som är viktigt är att försöka hitta så många objekt som möjligt och sedan att varje objekt ansvarar så mycket som möjligt för operationer på sig självt. När man har bestämt sig för objekten kan man sedan skriva klasser som beskriver dem. Detta säger att varje objekt tillhör en klass. Man kan säga en klass är en typ.
En klass är endast en definition ett objekt och inte det riktiga objektet. Vi definierar objekt med hjälp av att skriva en klass och sen använda klassen som mall för att skapa ett eller många objekt av klassen. Detta gör man genom man deklarerar en variabel med typen av klassen och sen använda nyckelordet **new** för att skapa objektet. När man har ett objekt, kan man sen använda dess tjänster för att lösa problem.

```
 8  public class GuessNumbersProg
 9  {
10      public static void Main(string[] args)
11      {
12          //1. Declare an instance of the class GuessNumber
13          PlayNumbers game;
14
15          //2. Create the instance (the game object)
16          //PlayNumbers() is acall to constructor of the PlayNumbers class
17          game = new PlayNumbers(); //PlayNumbers() is acall to constructor
18
19          //3. Use the services of the game object
20          game.Play();  //start the game
21
22          //All done, just let the program window be waiting
23          //normally ReadLine has return value but we don't need to use it here
24          Console.ReadLine();
25      }
26  }
27
```

I exemplet här, deklareras en instansvariabel som kommer att hålla ett objekt av klassen **PlayNumbers** på rad 13. Det riktiga objektet skapas på rad 17 och den kommer att fungera som dess klass, **PlayNumbers** har bestämt. På rad 20 används en tjänst som är programmerad i klassen, nämligen **Play**. Denna tjänst eller rättare sagt, metod, utför sitt jobb som den är instruerad i hur i klassen och kan använda sig av de värden som är sparade i objektet för vilka klassen.

En klass är en mall som kompilatorn använder för att skapa instruktioner vid kompilering för **CLR** (Common Language Runtime) för att sedan kunna skapa objekt under körningen. En klass främsta uppgift är att definiera:

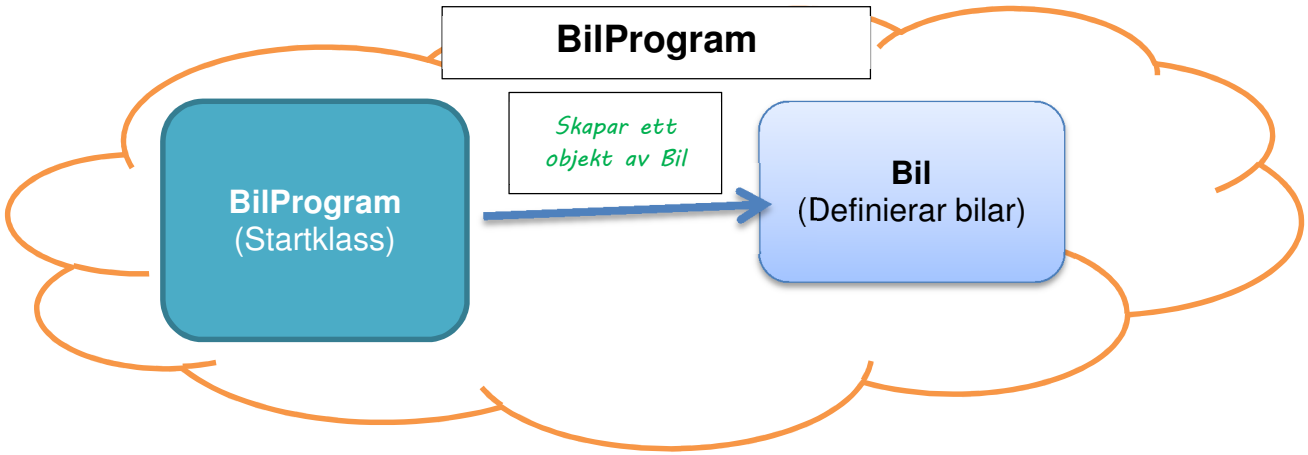
- fältvariabler, behållare för att spara värden dess objekt ska lagra, d.v.s. definiera variabler som kallas för **fält, attribut, instansvariabler, medlemsvariabler,**
- de operationer den ska utföra, d.v.s. definiera så kallade ”**Properties**” (egenskaper) och **metoder**. En Property är en speciell typ metod som behandlas i ett senare kapitel.

En klass är alltså bara en definition som beskriver objektets tillstånd (variabler) och objektets beteende (metoder). Varje klass skall ansvara och behandla endast det den är avsedd för. Den ska inte göra jobbet som tillhör en annan klass men den kan kollaborera med andra klasser. En klass skall erbjuda tjänster till andra klasser och använda tjänster från andra klasser. En **Bil** klass skall bara hantera bilar, deras färg, registreringsnummer och pris.

En klass kan även bygga sig på objekt av andra klasser. En klass Person kan till exempel använda ett objekt av klassen Adress som sitt fält, för att undvika att själv skriva kod för hantering av ett Person objekts adress. **Bil** klassen kan ha några objekt av **Hjul**. Allt som berör hjul skall programmeras i Hjul klassen. Bil-klassen kan då använda sig av hjulklassens färdigt programmerade tjänster (metoder).

Klasser och deras objekt i programmering är mycket duktigare än de objekt vi är vana att använda i verkligheten. Ett objekt av klassen **Bil**, kan gasa, bromsa, byta färg och även skrota sig själv.

Som en mjukstart och för att lära oss med C# språkets grunder börjar vi med enkla konsolapplikationer där vi skriver två klasser, en klass som vi kan i våra fortsatta diskussioner referera till som startklass och en som genomför en specifik uppgift. Startklassen är den som innehåller en special metod som kallas för **Main**. Denna metod kommer att automatiskt köras när programmet startas av operativsystemet, och all kod som finns i metoden kommer att köras sats för sats.



3. Delar av en klass

En typisk klass består av ett antal medlemmar. Vi har börjat bekanta oss med instansvariabler och metoder så här långt. De andra begrepp som listas i koden nedan kommer vi att studera under kursensgång. Kodskeletten representerar i princip en mall och en typ checklista för att komma ihåg varje gång du skriver en klass.

Fältvariabler används för att lagra data för de attribut som beskriver ett objekt av en klass. Val av attribut beror på avsikten med klassen. För en bil kan det vara t.ex. färgen, antal hästkrafter, modellen och för en produkt blir namnet, priset och kategorin som kan vara attribut. Valet beror på syftet med programmet. En produktklass kan användas för att beskriva en fysisk sak som t.ex. en banan, en mikrovågsugn där man är intresserad av färgen, vikten, måtten, tillverkaren, men om ändamålet med en produktklass är att skriva ett kvitto vid försäljning av varan, så är andra egenskaper hos produkten som blir intressanta, t.ex. priset, antalet och momssatsen.

```
namespace ettGruppNamn
{
    public class KlassNamn  //klassens namn och synlighet
    {
        //1. fältvariabler  (kallas också instansvariabler för medlemsvariabler och attribut)
        //2. Konstruktorer med eller utan arameterar (speciala metoder)

        //3. Properties, get för läs-,och set för skrivrättigheter

        //4. Metoder med eller utan parametrar, med eller utan retur-typ


    }//Klassens slutmarkering
}//slutmarkering för namespace
```

Klassnamnet måste börja men bokstav, inte kan vara ett reserverat ord, inte kan innehåller mellanslag och andra sådana symboler förutom understryckningstecknet. Substantiv eller kombination av substantiv brukar väljas för klassnamn.

Fältvariabler har samma regler som klassnamnet och de som används för att spara värden (t.ex. modellen, priset och antal dörrar för ett objekt av Bil) för varje objekt av klassen, eftersom varje objekt är ett unikt exemplar. Varje objekt har sin egen uppsättning av värden sparade i objektets instansvariabler. Din bil och kompisens bil även om de är exakt likadana är två unika objekt, unika i hela universum. Fältvariabler måste deklareras och vid deklarationen anger man synlighet (åtkomligheten, t ex private), typen (string, double, Hjul, osv) och ett namn (modell, pris, etc.). Namnet är egentligen alias till adressen av det minnesutrymme som reserverats för värdet under körningen. Exempel på deklarationer av fält i en klass:

```
//Denna rad är en kommentar och
using System;  // detta säger till kompilatorn att vi använder klasser från System (Console)
namespace FordonApplikation //samma i alla filer
{
 public class Bil
 {
    //Del 1: instansvariabler - var sparsam!
    //instansvariabler (attribut) som beskriver en bil (vilken som helst)
    //beskriver objektets tillstånd, objekt är en instans av denna klass
    private string modell;
    private int antalDörrar;
    private double pris;
    private bool besiktad;

 //resten av klassen

 }
}
```

Varje objekt har sin uppsättning av klassens variabler. Om det skapas 300 Product objekt, finns det 300 uppsättningar av värden, en för varje objekt; varje objekt har då sitt eget produktnamn, styckepris och modell. Alla metoder i klassen har access till variablerna. Variabler som deklareras private kan inte nås utanför klassen. Åtkomst till värden sparade i privata variabler kan regleras via **Properties**, något som vi kommer att diskutera senare i kursen. Variabler som deklareras protected fungerar som private men med undantag av att även subklasser nå dem. Med subklass

menas en klass som ärver en annan klass och det är också något som vi avvaktar till senare i kursen att prata om. Variabler som deklareras public nås utan någon begränsning av alla klasser i samma och andra applikationer.

I objektorienteringens värld strävar man efter att undvika publika variabler och i stället kapsla in och gömma så mycket som möjligt av data och operationer i klassen. Tumregeln är att alltid deklarera instansvariablerna i en klass som private, ibland protected men aldrig public. När det gäller metoder i en klass så behöver en del av metoderna vara publika för att kunna synas och användas av andra klasser. Dessa ingår då i klassens gränssnitt för kommunikation med andra klasser. Däremot bör de metoder som används internt i en klass deklareras som private. Anledningen till att man vill gömma saker är att man enklare kan säkerställa att programmet fungerar som det är tänkt och avsiktlig eller oavsiktlig felanvändning minskas.

Konstruktorn är en viktig del av en klass. Den är en special metod som heter samma som sin klass och har ingen returtyp (void, int ,double). En klass konstruktor körs alltid direkt när ett objekt av klassen skapas i applikationen med nyckelordet new. Om vi inte skriver någon konstruktor i en klass så skapar kompilatorn en parameterlös konstruktor (default konstruktor) vid kompileringen och bakar in den i assemblyn. Så här ser ut en s.k. default konstruktor:

```
public Bil()
{
     modell = "";       //modell ej angivet
     antalDörrar = 4;   //standard
     pris = 0.0;        //priset ej känt från början
     besiktad = false; //inte besiktad (initiering)
}
```

Properties kan används för att ge access till privata variabler i ett objekt. För läsaccess används get- och för skrivaccess används set-metoderna. Koden till höger visar ett exempel där både set och get metoderna har definierats. En Property anropas som om den är en variabel. **OBS**. Avsikten här är inte att du ska förstå koden nu, utan bara för att visa hur det ser ut.

```
minBil.AntalDörrar =  3;  //set property anropas
Console.Writeline (minBil.antalDörrar);  //get property
```

```
/// <summary>
///  Property för att ge både läs- och skrivaccess till
///  värdet av den privata variabeln antalDörar.
///  value är ett reserverat ord som automatiskt får samma
///  typ som Propertyn, här blir den int.
///  value innehåller ett nytt värde för antalDörrar.
/// </summary>
public int AntalDörrar
{
    get
    {
        return antalDörrar;
    }
    set
    {
        antalDörrar = value;
    }

}
```

Metoder är det som gör klassen intelligent och användbar. Samtliga uppgifter som en klass förväntas utföra måste skrivas i en metod. Det går alltså inte att skriva en sats löst i en klass. Bara deklarationer av instansvariabler kodas utanför metoder i en klass, allt annat måste tillhöra en metod. Metoder kan vara antingen en `void` metod eller ha en returtyp. En metod har en signatur, d.v.s. ett huvud och en kropp.
Signaturen börjar med en accessmodifierare, public, private eller protected, följt av en returtyp (void eller en datatyp), ett metodnamn och ett par parenteser. Inne i parentesen kan man deklarera variabler som kallas för argument eller metodparametrar. Parametrarna används för ta emot indata till metoden.

Metoder som nämndes kan vara antingen en `void` metod eller en metod som returnerar ett värde av en viss typ. Med returnera menas att skicka tillbaka ett värde till det ställe som anropade metoden. En `void` metod är en metod som utför sina instruktioner varje gång metoden anropas (d.v.s. aktiveras) men den returnerar inget värde. Här är ett exempel på en `void` metod som har fått uppgiften att fråga användaren om information om bilobjektet är besiktad eller ej. Den lagrar svaret som ett logiskt värde `true` (om beskiktad) eller `false` (om inte beskiktad) i den instansvariabel som är avsedd för det.

```
private void LäsInfoOmBesiktning()
{
    Console.WriteLine("Är bilen besikted (j/n)?");
    //Läs ett tecken från användaren och spara det i en temporär variabel (svar)

    char svar = char.Parse(Console.ReadLine());

    //Om användaren skriver j eller J är bilen besiktad; annars inte
    //== är likhets symbol (om vänstra del är lika med högra del)
    if ( (svar == 'j') || (svar == 'J') )
        besiktad = true;
    else
        besiktad = false;
}
```

1

2

```
48  //Läs indata och printa ut allt om bilen
49  public void LäsInputOchSkrivInfo()
50  {
51      //1. Fråga användaren om bilen är besiktad och spara svaret
52      LäsInfoOmBesiktning();
53
54      //2. Skriv all info som är sparad för bilen än så länge
55      VisaBilInfo();
56
57  }
```

När man vill använda denna metod (bilden ovan), skriver man bara namnet på metoden, om metoden anropas i sin egen klass som rad nummer 52 i skärmbilden här. Observera parenteserna och semikolontecknet!

Om en metod anropas från en annan klass, måste objektnamnet följt av en punkt och sen metodnamnet användas.

```
nyBil.VisaInfo();
```

När metoden LäsInfoOmBeskiktning anropas (rad 52), hoppar exekvering till början av den **LäsInfoOmBeskiktning** metoden (längst upp till vänster) och när denna metod har kört färdigt så kommer exekveringen tillbaka och forsätter med nästa sats (rad 55).

En metod kan utföra sin uppgift men till skillnad från en **void** metod kan den också returnera ett värde. Värdet kan vara ett beräkningsresultat eller ett true/false värde som berättar någon status, exempelvis om en inläsning eller en beräkning har gått bra (true) eller inte (false). Å andra sidan kan en metod också förvänta sig värden som input via sina argument. I båda fallen så måste typen av värdet bestämmas av metoden som visas i exemplet som följer.

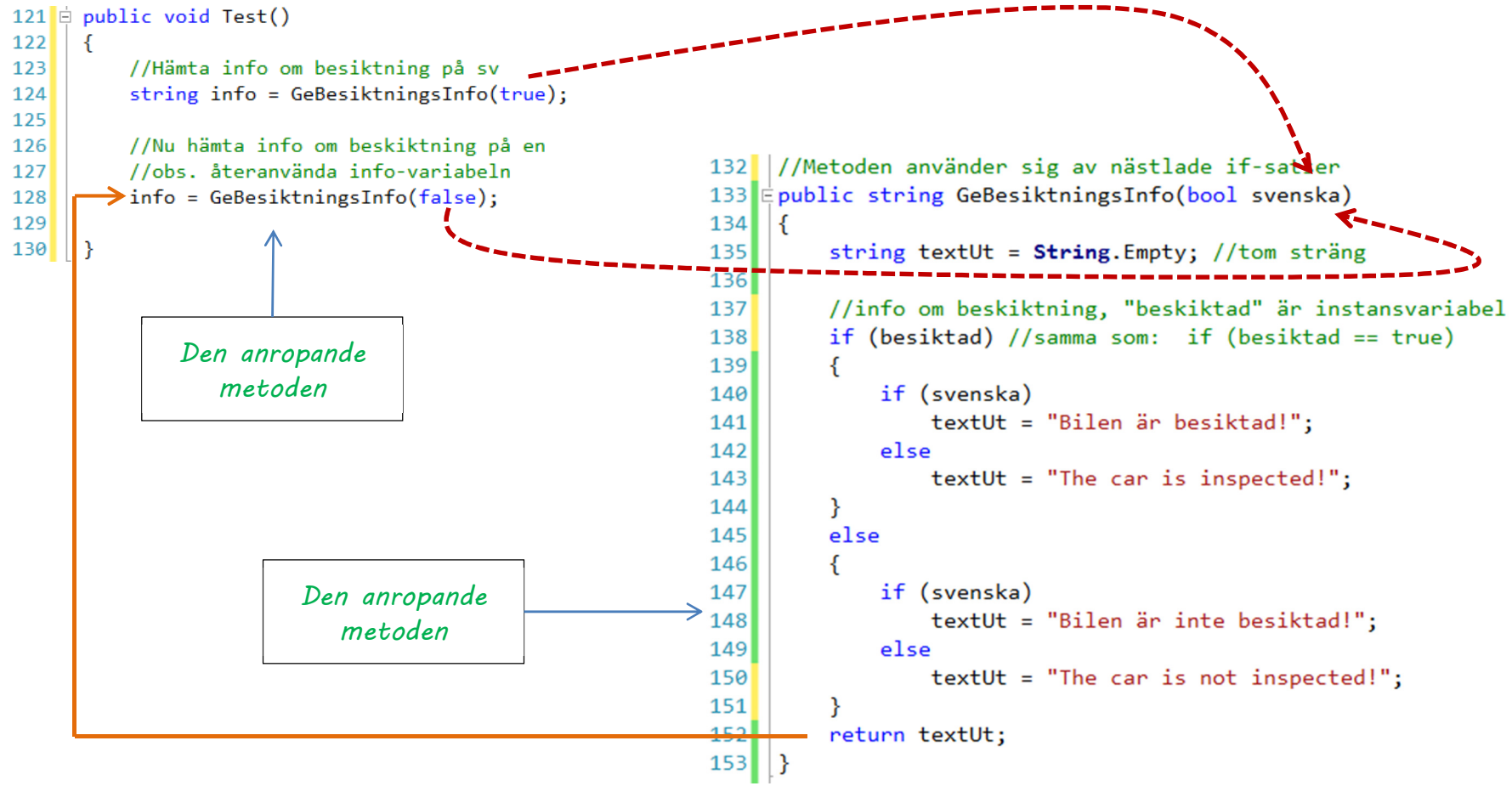
Första anropet vid rad 124 i metoden på bilden till vänster (nästa sida) gör att en av raderna 141 eller 148 i bilden på nästa sida exekveras, beroende på värdet lagrat i variabeln **besiktad**. Resultatet (texten) lagras i metoden i variabeln **textUt**. Notera speciellt att **textUt** endast är synlig inne i metoden. Dessutom har den ett kort liv, den lever endast under den tid metoden exekveras.

Nästa gång när metoden anropas (som t.ex. på rad 128 bilden nästa sida), skapas och initieras variabeln **textUt** på nytt. I stället för ett konstant true värde, går det också bra att skicka en variabel (rad 124 bilden till vänster på nästa sida).

När metoden **GeBeskiktningsInfo** är exekverad till rad nummer 152, skickas innehållet i variabeln **textUt** till den anropande metoden Test, och hamnar i variabeln info (rad 124). Detta beskriver en skillnad mellan anrop av en void metod och en metod med returtyp. Jämför metodanropet i det föregående exemplet på rad 52 (och rad 55) i den nedersta bilden med anropet på rad 124 (eller 128) på bilden nedan (t.v.). En void metod anropas utan något variabel att ta emot returvärdet. Det går också bra att anropa en metod med returtyp som en void metod men då har man förkastat returvärdet.

```
string input = Console.ReadLIne();      //här används returvärdet (en rad från konsolfönstret)
Console.ReadLine();                     //här förkastas returvärdet (oavsett vad användaren matar in).
```

Ordet funktion (**function**) förekommer också ofta som ett alternativt namn till metod. En void metod är då en funktion som inte returnerar ett värde.




```
121 public void Test()
122 {
123     //Hämta info om besiktning på sv
124     string info = GeBesiktningsInfo(true);
125
126     //Nu hämta info om beskiktning på en
127     //obs. återanvända info-variabeln
128     info = GeBesiktningsInfo(false);
129
130 }
```

```
132 //Metoden använder sig av nästlade if-satser
133 public string GeBesiktningsInfo(bool svenska)
134 {
135     string textUt = String.Empty; //tom sträng
136
137     //info om beskiktning, "beskiktad" är instansvariabel
138     if (besiktad) //samma som:  if (besiktad == true)
139     {
140         if (svenska)
141             textUt = "Bilen är besiktad!";
142         else
143             textUt = "The car is inspected!";
144     }
145     else
146     {
147         if (svenska)
148             textUt = "Bilen är inte besiktad!";
149         else
150             textUt = "The car is not inspected!";
151     }
152     return textUt;
153 }
```

4. Sammanfattning

En klass består huvudsakligen av två delar, instansvariabler och metoder. Instansvariabler lagrar ett objekts tillstånd medan metoder kan utföra operationer och därmed definierar objektets beteende.

En klass är en definition, en mall för att skapa objekt av en viss typ. Varje objekt måste tillhöra en klass, det vill säga vara av en viss typ. Ett objekt skapas med nyckelordet new. Då anropas klassens konstruktor som är en special metod som alltid när objektet skapas. Konstruktorn används för att initiera objektets tillstånd, d.v.s. tilldela objektets instansvariabler initiala värden. Den används också för andra initieringar om så behövs.

Det finns två typer av standard metoder, **void** och metoder med returtyp. En metod med returtyp utför ett antal instruktioner och returnerar ett värde till den metod som har anropat den. En **void** metod jobbar på samma sätt men den returnerar inget värde. Ett värde returneras med användning av nyckelordet **return**. Det är viktigt att komma ihåg att nyckelordet **return** också avbryter exekvering av metoden. All kod som skrivs efter en **return** sats kommer aldrig att utföras. C# kompilatorn kommer att varna dig om det om du av misstag anger den felaktigt.

Lycka till.

Farid Naisan,
Kursansvarig och lärare