Campus Nyköping

PROGRAMMERING I .NET C#1 - 09

OOP1

Förra gången

- Agil utveckling – Betygsresonemang
- Lägesrapport
- Lite nytt
  - Ternary operator
  - Files
  - Code along
- Demo
  - Hotellet
  - Ritprogrammet
- Övningar

Idag

- Upprop
- Demo av övningar
- Objektorienterad programmering

Övningar

- Schackbräde och riskorn
  - Svaret är?
- Bankomathistorik
  - Använd List<string>
- Hotellet – spara bokningar
  - Demo
- Matrisövningar
  - Svar läggs upp på skolans portal
- Laddningsfunktion till ritprogrammet
  - Fortsätt jobba med…

Vad är ett objekt?

- Vår värld innehåller en massa "prylar".
- Varje sådan pryl är ett objekt.
- Det kan vara allt ifrån verkliga prylar, till digitala eller påhittade.
- Med objektorinterad programmering kan vi hantera dem som en egen entitet.




Vad är objektorienterad programmering?

- Objektorienterad programmering:
  - Som namnet antyder refererar Objektorienterad programmering eller OOP till språk som använder **objekt** i programmering.
  - Objektorienterad programmering syftar till att implementera verkliga enheter.
  - Det huvudsakliga syftet med OOP är att binda samman data och de funktioner som fungerar på dem, så att ingen annan del av koden kan komma åt dessa data förutom just den funktionen.
  - Det finns ett antal grundpelare inom OOP, såsom som **arv**, **inkapsling**, **polymorfism**

Glödlampa

- Hur fungerar en glödlampa?
  - Man trycker på en strömbrytare
  - En krets sluts som, via två elledningar, transporterar elektroner till lampan.
  - I lampan finns en lys-diod, som strömmen går igenom
  - El-strömmen ska ha följande specifikation:
    - 230 V
    - 50 Hz
    - Växelström
  - Lysdioden drivs på svagström, så en transformator behövs.

  - Fråga: behöver vi veta allt det här för att kunna tända en lampa?

- I det här exemplet är lampan objektet, och det enda vi behöver kunna är att trycka på en strömbrytare. Allt annat är vanligtvis "gömt" för oss.

OOP

- Objektorienterad programmering (OOP) är ett programmeringsparadigm som låter dig paketera samman data och funktionalitet för att ändra dessa data, samtidigt som detaljerna döljs undan (som med glödlampan). Som ett resultat är kod med OOP design **flexibel, modulär och abstrakt**. Det gör det särskilt användbart när du skapar större program.

(**Paradigm** betyder modell, förebild, mönster eller mönstergillt exempel)

Fördelar med OOP

- OOP är snabbare och enklare att utföra
- OOP ger en tydlig struktur för programmen
- OOP hjälper till att hålla C #-koden DRY - "Don´t repeat yourself" och gör koden lättare att underhålla, ändra och felsöka
- OOP gör det möjligt att skapa återanvändbara applikationer med mindre kod och kortare utvecklingstid

Objektorienterad programmering

- Begrepp
  - Klasser — Classes
  - Fält/attribut — Fields/attributes
  - Egenskaper — Properties
  - Konstruktor — Constructor
  - Access Modifier — Åtkomstmodifierare

Metoder

Metod med både ut- och indata

```
static int Subtrahera(int a, int b)
 {
     int diff = a - b;
     return diff;
 }
```

Används med: `int resultatet = Subtrahera(25, 13);`

Klasser

- En klass används i objektorienterad programmering för att beskriva ett eller flera objekt. Den fungerar som en mall för att skapa, eller instansiera, specifika objekt inom ett program.
- Ett sätt att gruppera olika saker.

[access modifier] class [identifier]

Exempel:

```
public class Customer
{
        //All kod som beskriver och hanterar en Customer finns här
        // Fields, properties, methods and events go here...
}
```

En klass vi redan använder

```
class Program
{
        static void Main(string[] args)
        {
          ...vårt konsolprogram


        }
}
```

Klasser och objekt

- Klasser beskriver en samling av attribut och metoder
- Klassen Program
  - Metoden Main()

- Klassen Fruits
  - Attributet Type
  - Metoden MakeFruitsallad()

Objektet Banan
Objektet Apelsin
Objektet Päron

- Klassen Car
  - Attributet CarBrand
  - Metoden StartCar()

Objektet Volvo
Objektet Toyota
Objektet Tesla

C# och klasser

- Allt i C# är associerat med klasser och objekt, tillsammans med dess attribut och metoder. Till exempel: i verkligheten är en katt ett objekt. Katten har attribut, som vikt, antal ben, ras och färg, och metoder, som springa och sova.
- En klass är som en objektkonstruktör eller en "ritning" för att skapa objekt.

Attributes - Attribut

- Alla objekt har någon form av attribut.
- En bil kan t ex ha följande egenskaper
  - Färg
  - Ålder
  - Antal hjul
  - Hel eller trasig
  - Registreringsskylt
  - Ägare
  - Namn
  - Märke

Klasser och attribut

- Hittills har vi enbart haft metoder I våra klasser, mest för att dela upp vårt program I mindre delar

- En klass kan dock innehålla många fler saker:
  - **Attributes eller Fields**
  - Properties
  - Methods (Detta har vi redan provat)
  - Constructor

En enkel klass med ett attribute/fält

```
internal class Fruit
{
  // Åtkomsten är public, så vi kommer åt den från andra klasser, t ex Program
  public string type = “Apple”;

 // Här nedanför kan vi stoppa t ex metoder, som vi gjort innan
}
```

Datatyper

- Vi har hittils arbetat med primitiva datatyper
  - int, string, double, ulong, bool…
- När vi arbetar med klasser och object så använder vi istället klassens namn som datatyp
  - Cat, Fruit, Car…

Använda en klass

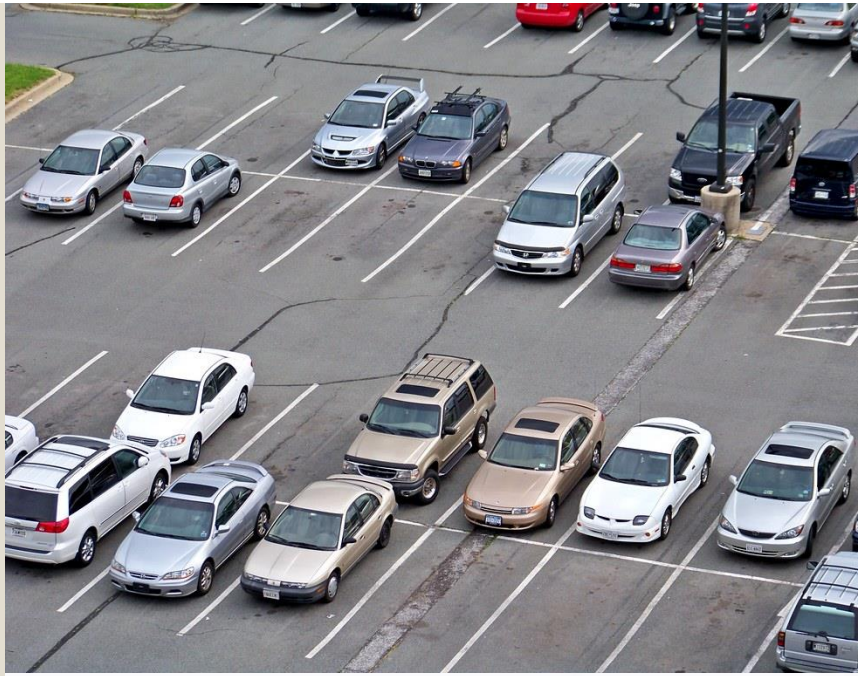
```
// Först skapar vi en ny frukt (en instans av Klassen Fruit)
Fruit f = new Fruit();

// Sedan läser vi en av fruktens variabler (ett attribut som Fruit har)
String type = f.type;

// Sist skriver vi ut attributet I konsollen.
 Console.WriteLine(type);
// Detta skriver ut Apple
```

new vs static

- Hittills har vi använt våra klasser en och en, d v s vi har inte behövt flera "versioner" av samma klass.
- Då behöver vi inte instansiera, och anger det med nyckelordet **static**.
- Men om vi behöver många instanser av ett object, t ex flera bilar eller flera frukter, så måste vi skapa en ny "version" av objektet, d v s instansiera med nyckelordet **new**.
- Då kan vi ha en hel parkeringsplats eller en stor fruktskål…



Åtkomst - Access

- **public: The type or member can be accessed by any other code in the same assembly or another assembly that references it.**
- private: The type or member can be accessed only by code in the same class or struct.
- protected: The type or member can be accessed only by code in the same class, or in a class that is derived from that class.
- internal: The type or member can be accessed by any code in the same assembly, but not from another assembly.
- protected internal: The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly.
- private protected: The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.
- Access Modifiers - C# Programming Guide | Microsoft Docs

Hela exemplet

- Program.cs

```
internal class Program
{
  static void Main(string[] args)
  {
     Fruit fruit = new Fruit();
     Console.WriteLine(fruit.type);
  }
}
```

- Fruit.cs

```
internal class Fruit
{
  public string type = "Apple";
}
```

Sätta ett värde på attribute senare

- Program.cs

```
internal class Program
{
  static void Main(string[] args)
  {
     Fruit fruit = new Fruit();
     fruit.type = “Banan”
     Console.WriteLine(fruit.type);
  }
}
```

- Fruit.cs

```
internal class Fruit
{
  public string type;
}
```

Skapa flera objekt

- Det är enkelt att skapa nya object.

```
Fruit fruit1 = new Fruit();
Fruit fruit2 = new Fruit();
```

Eller I en loop:

```
for (int i = 0; i < 10; i++)
{
        Fruit fruit = new Fruit();
        ...
}
```

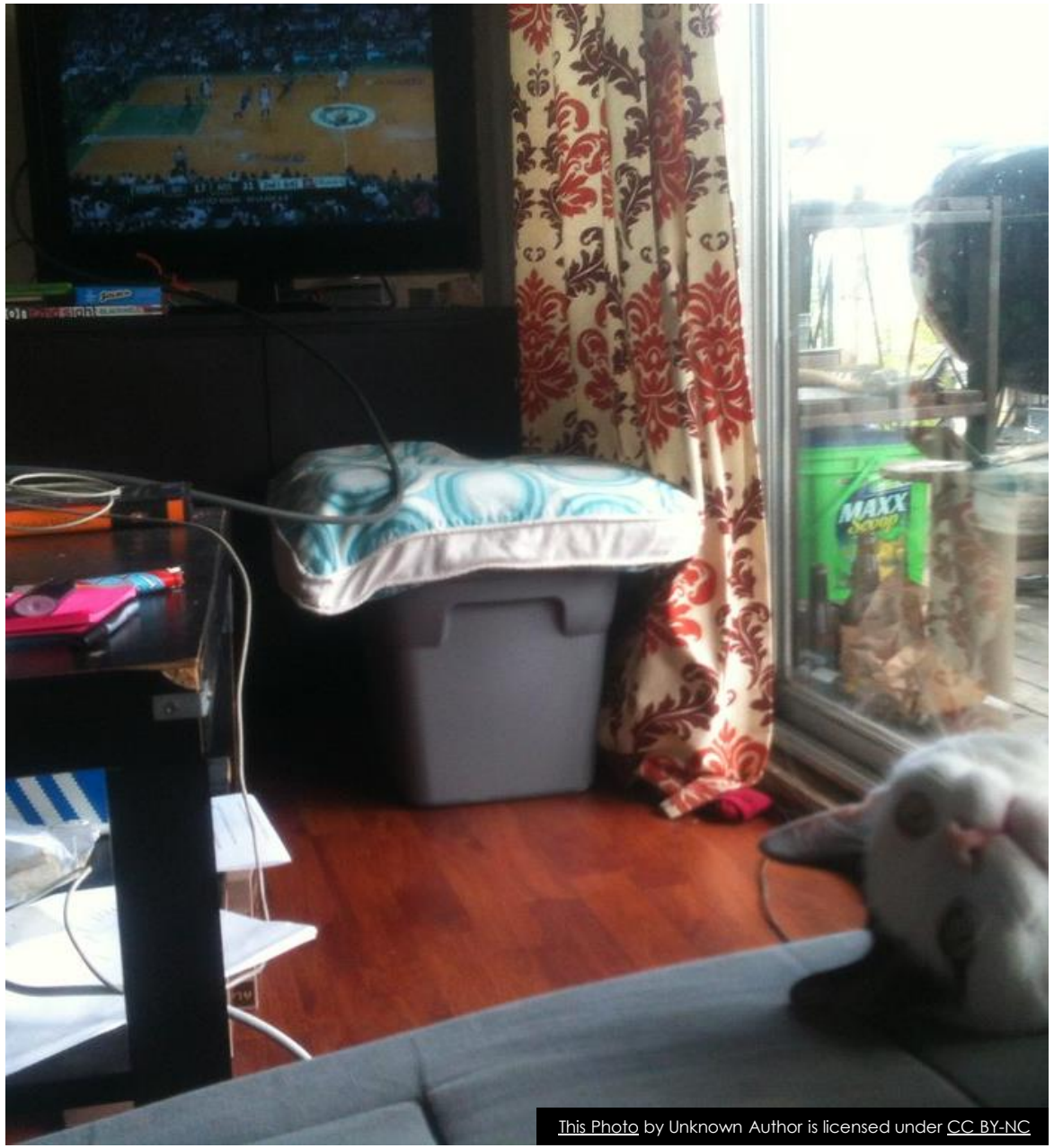
Klasser och metoder

- Eftersom en class (ett object) är som vilken datatyp som helst, kan vi även använda dem som inparametrar och returvärde I en metod.

```
public static Fruit CalculateWeight(Fruit fruit)
{
      fruit.weight = 0.23
      return fruit;
}
```

Demo: OOP1

- Klasser
- Instanser
- Objekt
- Attribut/Fält
- Kod:
  - AttributeBasicDemo
  - PersonsWithCars

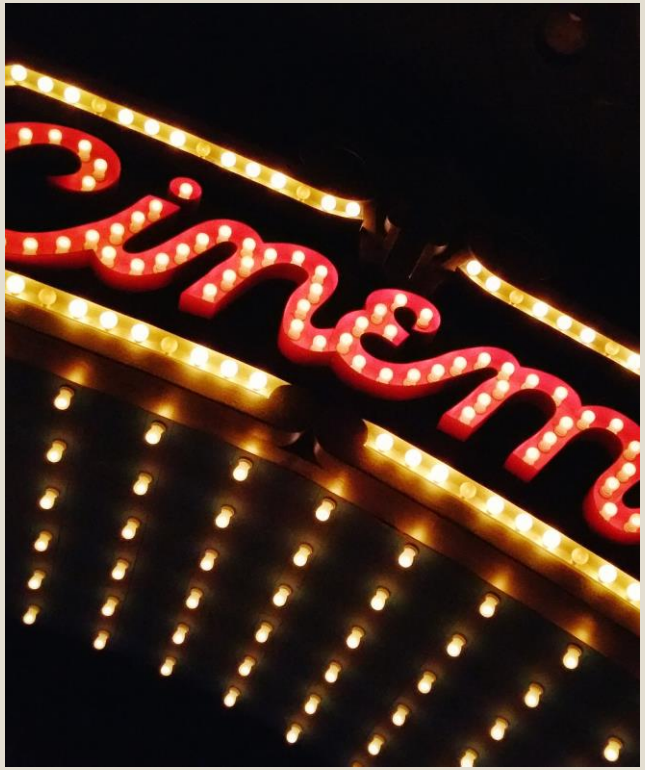


Övning 1 – Mina prylar

- Fundera ut tre olika "objekt" I din vardag.
  - Det kan t ex vara din dator, en person I familjen, något husdjur, din bil…
  - En av objekten ska du ha mer än en av, och den ska hanteras som en list.
- Skapa en klass med cirka fem attribute, som passar in på ditt objekt
  - Det kan vara namn, storlek, färg, vikt…
- Bygg en app där du fyller dina object med information.
  - Prova både att hård-koda, och att mata in via ReadLine.
- Visa informationen I ett överskådligt format I konsollen.

Övning 2 - Filmlistan

- Skapa en klass: Movie
- Lägg in ett antal properties:
  - Name (str)
  - Director(str)
  - ReleaseDate (Datetime)
  - LengthInMinutes (int)
  - SuitableForChildren (bool)
  - Stars(str[])
- Skapa minst tre instanser av en film, med relevant data (se till att du har med någon film lämplig för barn)
- Skapa en metod som tar var och en av filmerna, och skapar en sträng som ser ut såhär:
  - [Name] är en film som regisserats av <director> med premiär <Date> och är <LengthinMinutes>.
  - Skådespelarna I filmen är <Stars>. Den är <SuitableForChildren ? “” : “inte”> lämplig för barn.



Övning 2: Filmlistan forts.

- Dela upp din kod I så små delar du kan
  - Sagan ska byggas av en separat metod.
  - Sagan ska skrivas ut av en separat metod.
  - Ålder ska räknas ut av en separat metod
  - Antal ben – ska byggas av en separat metod

Gamla övningar

- Fortsätt jobba med de gamla övningarna, om du inte är klar.
  - Schackbräde och riskorn
    - Svaret är?
  - Bankomathistorik
    - Använd List<string>
  - Hotellet – spara bokningar
    - Lösningsförslag idag
  - Matrisövningar
    - Svar läggs upp på skolans portal
  - Laddningsfunktion till ritprogrammet
    - Fortsätt jobba med…

Länkar

- https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/
- https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/object-oriented-programming

- C# OOP (Object-Oriented Programming) (w3schools.com)
- C# Classes and Objects (w3schools.com)
- C# Multiple Classes and Objects (w3schools.com)