PROGRAMMERING I .NET C#1 - 17

OOP 4 : Inkapsling/Åtkomstmodifierare, Polymorfism, Abstract

Förra gången

- Blandat
  - List
    - Olika sätt att loopa
    - Lägga till och ta bort
  - Objekt och parametrar
    - Parametrar kan vara andra object, eller listor
  - Placera markören I konsollen

Idag

- Lägesrapport
- Mer objektorienterad programmering
  - De tre pelarna
    - Arv
    - Inkapsling
    - Polymorfism
  - Access modifiers
  - Method Overload
  - Virtual/Override
  - Abstract
  - Interface

Lägesrapport

- Programmeringens grunder
- C#/.NET Historik
- Visual studio
- Konsol-applikation
- Syntax och semantik
- Datatyper
- Variabler
  - Arrayer och Matriser
- Operatorer
- Strängar
- Booleans
- User Input
- Metoder
- Hur man debuggar
- Versionshantering/GIT

- GUI i konsolmiljö
- Filer och strömmar
- Objektorienterad programmering
- Klasser
- Properties
- Samlingar
- Kod-principer för objektorientering
- Arv
- Inkapsling
- Polymorfism
- Åtkomstmodifierare
- (Ramverk och bibliotek)
- Betygsgrundande inlämningsuppgift

- 6 av 9 veckor.

Sista genomgångarna

- Tre veckor kvar
  - 6 lektionstillfällen

- Idag?
  - Access modifiers
  - Method Overload
  - Virtual/Override
  - Abstract
  - Interface

- Onsdag: VG-uppgiften
- Generics
- Andra kollektioner

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 objektorienterad 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**

Objektorienterad programmering

- Tre pelare:
  - **Inheritance – Arv**
    - Ett arv innebär att vi kan skapa klasser som ärver attribut (variabler) och metoder från någon annan klass. Vi kan på så sätt utgå från en befintlig konstruktion och bygga vidare på den.
  - **Encapsulation – Inkapsling**
    - Inkapsling kan man se som ett skydd för en klass så att den inte kan användas på fel sätt. Det kan också handla om att inte visa/exponera för många saker, speciellt inte sånt som inte är relevant för den som ska använda klassen.
  - **Polymorphism – Polymorfism**
    - Ordet polymorfism används i olika sammanhang och beskriver situationer där något förekommer i flera olika former. I datavetenskapen beskriver den konceptet att objekt av olika typer kan nås via samma gränssnitt. Polymorfism innebär också att flera olika subklasser under en basklass kan hanteras som om de vore instanser av basklassen

Object Oriented Programming

Encapsulation - Everybody should mind their own business

Inheritance - Yes, we're like our parents, but much more special

Polymorphism

- → To my mom, I'm a saint.
- → To my spouse, I'm a nuisance.
- → To my friends, I'm the Dungeon Master.

The 3 Pillars of Object-Oriented Programming (OOP) Brought Down to Earth - Tech Elevator

Inkapsling - Encapsulation

- Inkapsling, i samband med C#, refererar till ett objekts förmåga att dölja data och beteenden som inte är nödvändiga för användaren. Inkapsling gör det möjligt för en klass, egenskaper, metoder och andra medlemmar att betraktas som en enda enhet eller ett objekt.
- Följande är fördelarna med inkapsling:
  - Skydd av data från oavsiktlig korruption
  - Specifikation av tillgängligheten för var och en av medlemmarna i en klass, till koden utanför klassen
  - Koden blir mer flexibel, utbyggbar och minskad komplexitet
  - Mindre koppling mellan olika objekt och därmed förbättring av kodunderhållet
- Inkapsling används för att begränsa åtkomsten till medlemmarna i en klass för att förhindra att användaren av en viss klass manipulerar objekt på sätt som inte är avsedda av designern. Medan inkapsling döljer den interna implementeringen av klassens funktioner utan att påverka systemets övergripande funktion, tillåter den klassen att betjäna en begäran om funktionalitet och lägga till eller ändra sin interna struktur (data eller metoder) för att passa förändrade krav.
- Inkapsling är också känd som **information hiding**.

Varför Access modifiers

- Åtkomstmodifierare är nyckelord som definierar tillgängligheten för en medlem, klass eller datatyp i ett program. Dessa används främst för att begränsa oönskad datamanipulation av externa program eller klasser.
- Det ser också till att vi bara ser det vi behöver, på rätt plats.

Access Modifiers

Caller's location	**public**	protected internal	**protected**	**internal**	private protected	**private**
Inne I klassen	✔	✔	✔	✔	✔	✔
Ärvd klass (Samma assembly)	✔	✔	✔	✔	✔	✘
Icke ärvd klass (samma assembly)	✔	✔	✘	✔	✘	✘
Ärvd klass (annat assembly)	✔	✔	✔	✘	✘	✘
Icke ärvd klass(annat assembly)	✔	✘	✘	✘	✘	✘

public

- Public gör att klass/metod/property är tillgänglig precis överallt
- Det är inte fel att använda public, men man bör bara använda det där det behövs
- Även andra refererade projekt eller klassbibliotek kan komma åt allt.

internal

- Begränsar åtkomst från andra projekt/klassbibliotek
- Fungerar som public, om du bara har ett projekt, utan referenser till andra projekt
- Om du inte anger åtkomstmodifierare, så är klasser per default internal.

protected

- Ger tillgång till klass/metod/property i samma klass, och de klasser som ärver från samma klass
- Ser till att alla klasser som har med varandra att göra har tillgång till varandras klass/metod/property .

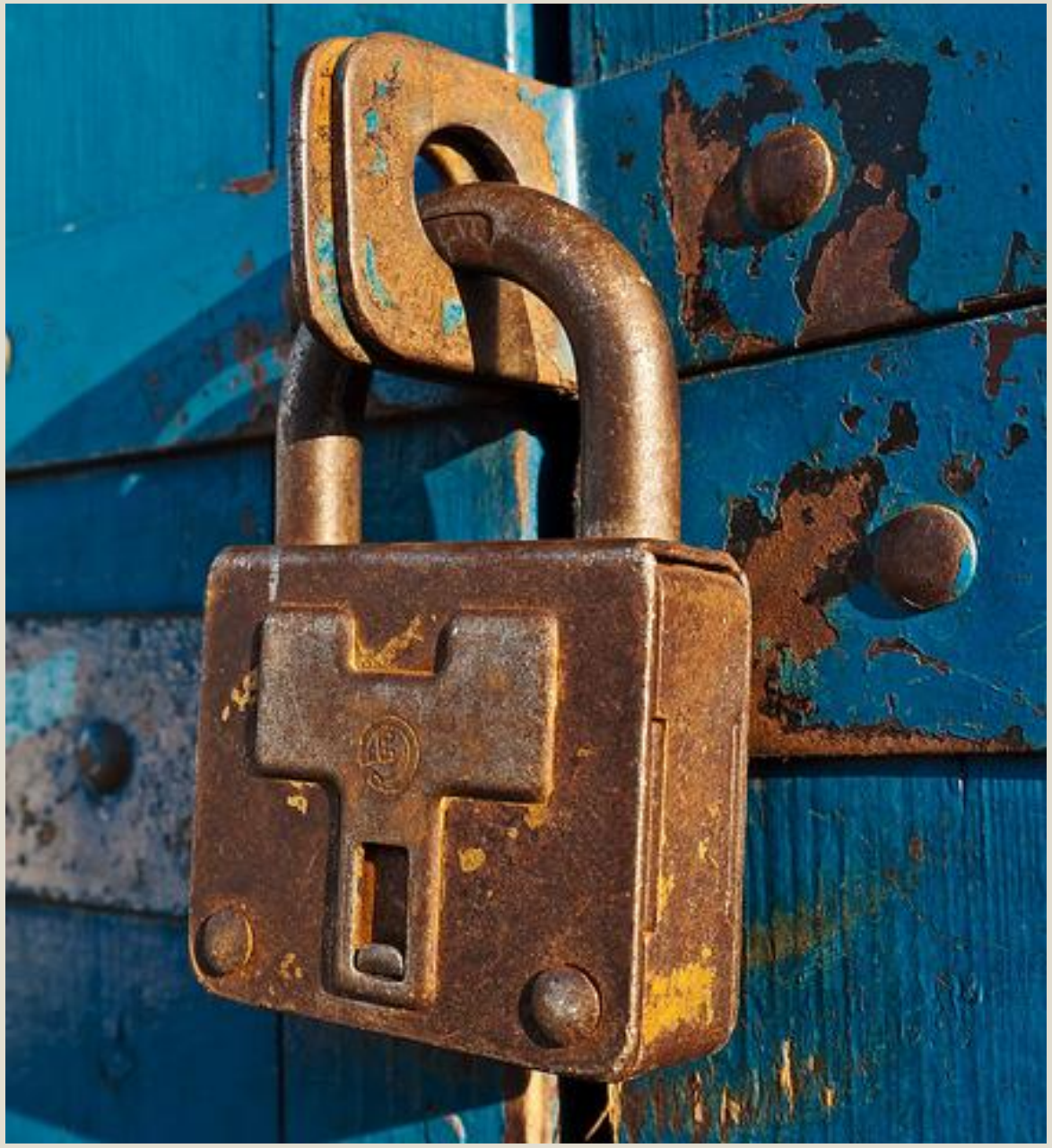
private

- Endast anrop inom samma klass är tillåtet

Råd

- Du kan välja två vägar då du kodar:
  - Sätt ALLTING till private, och ändra till den accessmodifierare som behövs, där det behövs.
  - Sätt allting till internal/public, och gå sedan igenom koden och refaktorera den.
    - Ställ frågan: Behöver en annan klass komma åt den här metoden, variabeln, attributet/propertyn?
  - Såhär I början föreslår jag, om du är osäker, att ändå sätta allt till internal/public, och gradvis övergå till private, under kursen.

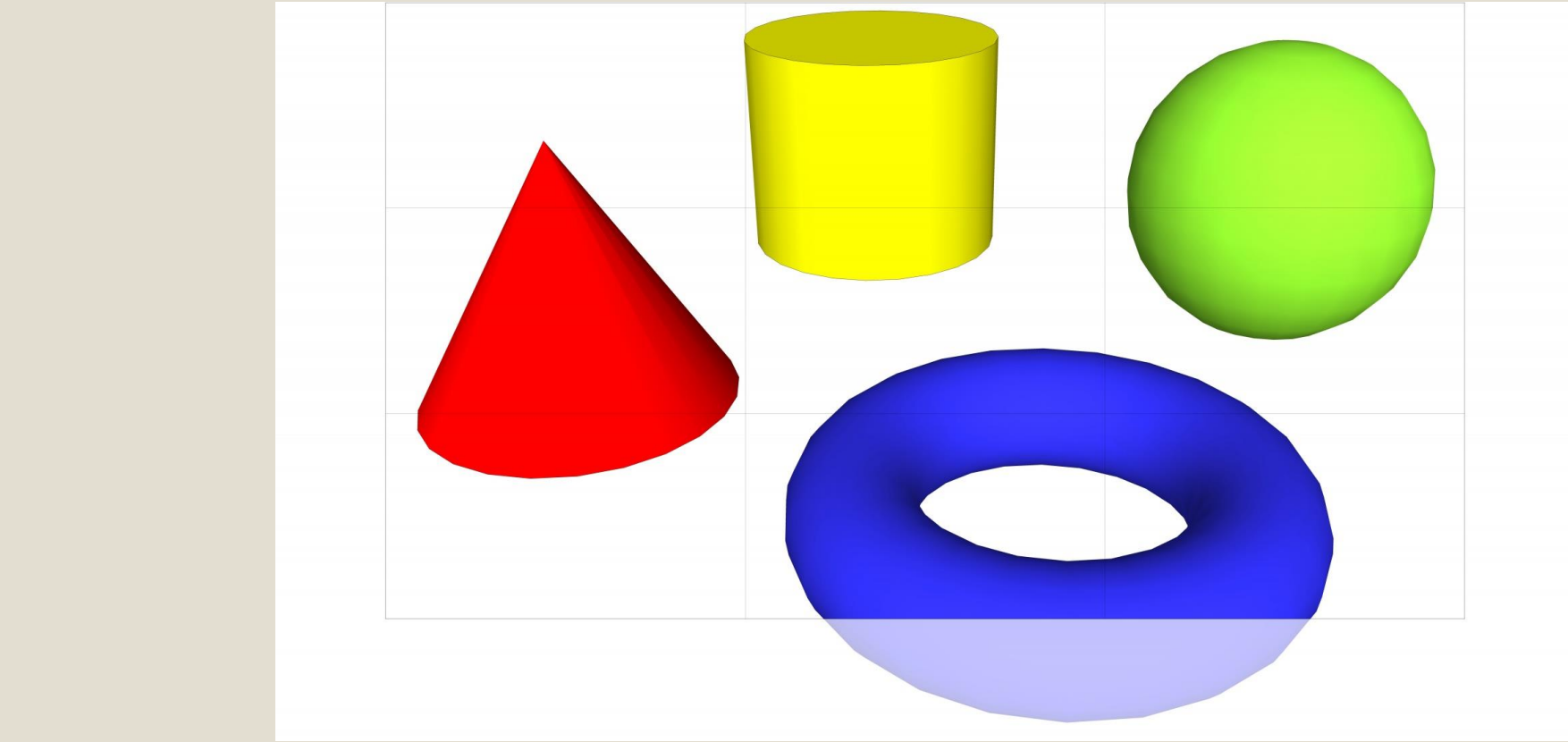
- Markera privata variable/attribute/properties med ett _ I början.



Code-Along

- AccessModifiers

Polymorfism - Polymorphism



Polymorfism

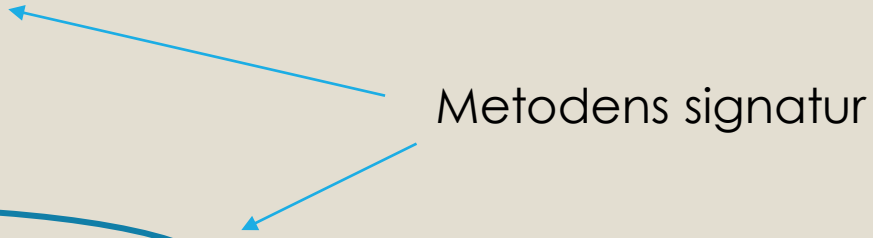
- Polymorfism betyder "många former", och det uppstår när vi har många klasser som är relaterade till varandra genom arv.
- Arv låter oss ärva properties och metoder från en annan klass. Polymorfism använder dessa metoder för att utföra olika uppgifter. Detta gör att vi kan utföra en enda åtgärd på olika sätt.
- Tänk till exempel på en basklass som heter Djur som har en metod som kallas animalSound(). Subklasser av Djur kan vara Grisar, Katter, Hundar, Fåglar - Och de har också sitt eget genomförande av ett djurljud (grisen säger oink, och katten säger mjau, etc.)
- Varför och när man ska använda "Arv" och "Polymorfism"?
  - Det är användbart för **återanvändning av kod**: återanvända properties och metoder för en befintlig klass när du skapar en ny klass.

Polymorfism - Overload

- Man kan ha metoder som har samma namn, så länge antalet inparametrar och/eller datatyperna för inparametrarna är olika.

- Exempel

```
public static int Multi(int a, int b)
{
  return a * b;
}
public static double Multi(double a, double b)
{
  return a  b;
}
```



Metoder: virtual och override

- Subklasser kan innehålla properties, constructors och methods
- För att kunna använda en metod I en subclass ska den först definieras I basklassen, med nyckleordet **virtual**
- Därefter kan vi göra en override I subklassen.
- En virtuell metod är en metod som sen kan omdefinieras i ärvda subklassen. I C#har en virtuell metod en implementering i en basklass samt även i den ärvda subklassen. Den används när en metods grundläggande funktionalitet är densamma men ibland behövs mer funktionalitet i den ärvda klassen.
- Basklassen:

```
public virtual void myMethod()
{
    ..här händer något.
}
```

- Subklassen

```
public override void myMethod()
{
   …här händer något annat…
}
```

- Virtual Method in C# (c-sharpcorner.com)

Properterties: Virtual och override

```
internal class BasKlassen
{
	public virtual string Name { get; set; };
}

internal class SubKlassen
{
	public override string Name { get; set; };
}
```

Abstraktion

- Dataabstraktion är processen att dölja vissa detaljer och visar endast väsentlig information för användaren.
- Nyckelordet Abstract används för klasser och metoder:
- Abstrakt klass: är en begränsad klass som inte kan användas för att skapa objekt (för att få tillgång till den måste den ärvas från en annan klass).
- Abstrakt metod: kan endast användas i en abstrakt klass, och den har inte något innehåll. Innehållet tillhandahålls istället av subklassen.
- Ordförklaring: Ett abstrakt är en kort sammanfattning av din *forskning*. Det är avsett att beskriva ditt arbete utan att gå in i detalj. Sammanfattningar ska vara fristående och koncisa och förklara ditt arbete så kort och tydligt som möjligt .
- Varför och när man ska använda abstrakta klasser och metoder?
  - För att uppnå säkerhet **- dölja vissa detaljer och bara visa de viktiga detaljerna i ett objekt.**

Interface

- Ett annat sätt att uppnå abstraktion i C#, är med Interface.
- Ett gränssnitt är en helt "abstrakt klass", som endast kan innehålla abstrakta metoder och egenskaper (med tomt innehåll):
- Det anses vara god praxis att börja med bokstaven ”I" i början av ett gränssnitt, eftersom det gör det lättare för dig själv och andra att komma ihåg att det är ett gränssnitt och inte en klass.
- Som standard är medlemmar i ett gränssnitt abstrakta och offentliga.
- För att få tillgång till gränssnittsmetoderna måste gränssnittet "implementeras" av en annan klass.
- Man kan se ett Interface som en ”ritning” av vad subklasserna måste innehålla.
- Varför Och när ska gränssnitten användas?
  - För att uppnå säkerhet - dölja vissa detaljer och bara visa de viktiga detaljerna i ett objekt (gränssnitt).

Länkar

- C# Inheritance (w3schools.com)
- The 3 Pillars of Object-Oriented Programming (OOP) Brought Down to Earth - Tech Elevator
- What is Encapsulation in C#? - Definition from Techopedia
- C# Access Modifiers (w3schools.com)
- C# Method Overloading (w3schools.com)
- https://www.w3schools.com/cs/cs_polymorphism.asp
- https://www.w3schools.com/cs/cs_abstract.asp
- C# Interface (w3schools.com)