Översikt

- Skapa klasshierarki.
- Utöka .NET Framework klasser.

.easec

Lektion 1: Skapa klasshierarki

- Vad är klass?
- Vad är arv?
- Klassen Beverage.
- Klassen Coffee.
- Kalla på klassmedlemmar.
- Skapa basklass -abstract.
- Abstract class –exempel.
- Skapa basklass –sealed.
- Skapa medlemmar i basklass –virtual.
- Access Modifiers för klassmedlemmar.
- Ärva från basklass.

.easec

Lektion 1: Skapa klasshierarki (forts.)

- Override på medlem i basklass.
- Override på medlem i basklass med new.
- Använda sealed.
- Kalla på basklass constructor och medlemmar.
- Kalla på basklass constructor från underklass.
- Kalla på basklass metod från underklass.

.easec

Vad är klass?

- Klass är en ritning (eng: blueprint), med ett antal metoder och egenskaper, som beskriver ett objekt.
- T ex klassen bil, som beskriver objektet bil, med ett antal egenskaper och ett antal metoder.
- Dessa kan ärvas nedåt i programkonstruktionen.

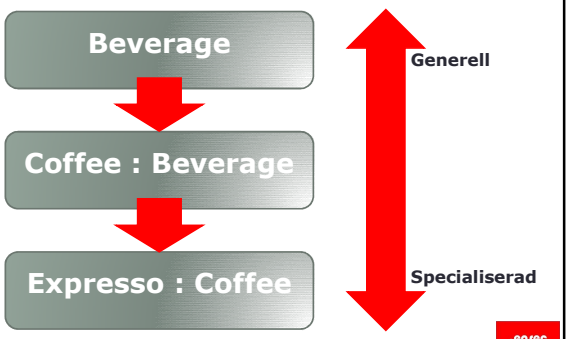
.easec

Vad är arv?

- Istället för att skapa nya klasser från start, kan du arbeta med befintliga klasser som bas för din nya klass.
- Arv ger möjlighet att bygga mer specialiserade klasser, nedåt i strukturen.
- Om din nya klass ärver från befintlig klass, kallas denna för derived class och den klass som den ärver ifrån kallas för base class.

.easec

Vad är arv? (forts.)



.easec

Klassen Beverage

```
public class Beverage
{
        protected int servingTemperature;
        public string Name { get; set; }
        public bool isFairTrade { get; set; }
        public int GetServingTemperature()
        {
                return servingTemperature;
        }
}
```

.easec

Klassen Coffee

```
public class Coffee : Beverage
{
        public string Bean { get; set; }
        public string Roast { get; set; }
        public string CountryOfOrigin { get; set; }
}
```

.easec

Kalla på klassmedlemmar

```
Coffee coffee1 = new Coffee();

// Använd medlemmar från basklass
coffee1.Name = ”Fourth Expresso”;
coffee1.isFairTrade = true;
int servingTemp = coffee1.GetServingTemperature();

// Använd medlemmar från underklass
coffee1.Bean = ”Arabica”;
coffee1.Roast = ”Dark”;
coffee1.CountryOfOrigin = ”Columbia”;
```

.easec

Skapa basklass –abstract

- När klass skapas, får du fundera på om du, eller andra skall använda klassen i ett arv.
- Om du vill skapa klass som skall vara bas för andra typer, dvs denna klass skall inte gå att accessas, skapa en klass av typen *abstract*.

```
abstract class Beverage
{
}
```

- Detta kan bero på att t ex att visa delar saknas, eller att du inte vill att klassen skall instansieras direkt.

.easec

Abstract class -exempel

```
abstract class Beverage
{
// Icke abstrakta medlemmar.
// Ärvda klasser kan använda dessa medlemmar utan att modifiera dem
        public string Name { get; set; }
        public bool IsFairTrade { get; set; }
        // Medlemmar som är abstract.
        // Ärvda klasser måste göra override och implementera dessa
        public abstract int GetServingTemperature();
}
```

.easec

Skapa basklass –sealed

- Om klass inte skall gå att ärva, skapa klassen med typen *sealed*.

```
public sealed class Tea : Beverage
{
}
```

- Du kan inte skapa klass som är både sealed och abstract, sealed är motsats till abstract.
- Sealed klass kan inte ärvas, medan abstract klass måste ärvas.

.eo/ec

Skapa medlemmar i basklass -virtual

- Om du har implementerat metod i din basklass och vill att klass som utgår från denna skall implementera egen metod – använd *virtual*.

```
public class Beverage
{
        protected int servingTemperature;
        public string Name { get; set; }
        public bool IsFairTrade { get; set; }
        public virtual int GetServingTemperature()
        {
                // Detta är standardmetoden.
                // Eftersom denna är deklarerad som virtual, kan
                // du göra override på denna i den ärvda klassen.
                return servingTemperature;
        }
}
```

.eo/ec

Access Modifiers för klassmedlemmar

Access Modifiers	Beskrivning
public	Medlem är tillgänglig för programkod i hela sammansättningen.
protected	Medlem är bara tillgänglig i denna klass, eller andra klasser som utgår från denna.
internal	Medlem är tillgänglig bara från intern programkod.
protected internal	Medlem är tillgänglig för intern kod och eventuella typer.
private	Tillgänglig inom samma klass.

.eo/ec

Ärva från basklass

- För att ärva från klass lägger du till : samt namnet för basklass.

```
public class Coffee : Beverage
{
}
```

- Underklass ärver alla medlemmar från basklass.
- Inuti underklass kan du definiera egna medlemmar för att bygga ut funktionalitet.
- Klass kan bara ärva från en basklass, men kan implemetera flera interface.

.easec

Override på medlem i basklass

- I några fall vill du förändra hur klassmedlem fungerar, i din underklass.

```
public class Beverage
{
        protected int servingTemperature;
        public virtual int GetServingTemperature()
        {
                return servingTemperature;
        }
}
```

- Eftersom metoden GetServingTemperature() är deklarerad som *virtual*, kan *override* användas i underklass.

.easec

Override på medlem i basklass (forts.)

```
public class Coffee : Beverage
{
        protected bool includeMilk;
        private int servingTempWithMilk;
        private int servingTempWithoutMilk;
        public override int GetServingTemperature()
        {
                if(includeMilk) return servingTempWithMilk
                else return servingTempWithoutMilk
;
        }
}
```

- Du kan använda samma metod för egenskaper, indexers och händelser.
- I alla fallen måste de vara markerade som virtual, för att det skall gå att göra override, ej constructors.

.easec

Override på medlem i basklass med new

- När nyckelordet *override* används, kommer den metod att **utöka** metod i basklass.
- När nyckelordet *new* används, kommer din metod att **gömma** metod i basklass.

```
public class Coffee : Beverage
{
        public new int GetServingTemperature()
        {
                // ...
        }
}
```

.eo/ec

Använda sealed

- Om du använder *sealed* på medlem du har gjort *override*, kommer du att tvinga eventuella klasser som ärver från din klass, att använda din implementering av klassmedlemmen och inte sin egen.

```
public class Coffee : Beverage
{
        sealed public override int GetServingTemperature()
        {
                // ...
        }
}
```

- Kan vara användbart när du vill ha kontroll över beteendet för din metod.

.eo/ec

Använda sealed (forts.)

- *Sealed* kan bara användas på medlem om denna medlem är *override*.
- Alla medlemmar är *sealed* om dessa inte är markerade som *virtual*.

.eo/ec

Kalla på basklass constructor och medlemmar

- Du kan använda nyckelordet *base*, för att få tillgång till metoder i basklass och constructors, från underklass.
- Metod i basklass:

```
base.GetServingTemperature();
```

- Constructor i basklass:

```
public Coffee(string name, bool isFairTrade, int temp)
  :base(name, isFairTrade, servingTemp)
```

.easec

Kalla på basklass constructor och medlemmar (forts.)

- Kan vara användbart i följande scenarios:
  - Du har gjort *override* på metod i basklass, men vill fortfarande använda funktionalitet i denna, tillsammans med din egna metods funktionalitet.
  - Du har skapat ny metod, men som en del av logiken för denna metod, vill du kalla på basklassens metod.
  - Du har skapat en *constructor* och som en del för initieringslogiken vill du kalla på basklassens constructor.

.easec

Kalla på basklass constructor från underklass

- Du kan inte göra overrride på *constructor* i underklass, när du skapar *constructor* i din underklass, kommer denna att först kalla på basklassens *constructor*, för att sedan köra eventuell logik du har lagt till i *constructor* för din underklass.
- *Constructor* kan även kalla på alternativ *constructor*, nyckelordet *base* anger vilken.

.easec

Kalla på basklass metod från underklass

- Du kan kalla på metod från basklass, antingen från kropp för metod eller egenskaper.

```
return base.GetServingTemperature();
```

.easec

Kalla på basklass metod från underklass (forts.)

```
public class Beverage
{
        protected int servingTemperature;
        public virtual int GetServingTemperature()
        {
                return servingTemperature;
        }
        // Constructor och ytterligare egenskaper visas inte.
}
public class Coffee : Beverage
        bool iced = false;
        protected int servingTempIced = 40;
        public override int GetServingTemperature()
        {
                if(iced)
                {
                        return servingTempIced;
                }
                else
                {
                        return base.GetServingTemperature();
                }
        }
}
```

.easec

Övning Kalla på construktor i basklass

.easec

Lektion 2: Utöka .NET Framework klasser

- Klasser i .NET Framework.
- Arv från .NET Framework.
- Utöka .NET Framework klass.
- Skapa skräddarsydd.
- Skräddarsydda klasser för undantag.
- Kasta undantag.
- Generic Class.
- Utöka funktionalitet.

Klasser i .NET Framework

- .NET Framework innehåller tusentals klasser som tillhandahåller bredda funktioner.
- När du skapar egna klasser bör du titta på dessa och basera dina egna klasser på dessa.
- Kortare utvecklingstid och höjer även kvalité på progamkod.
- Standardiserad funktionalitet.

Arv från .NET Framework

- Finns även möjlighet att bygga ut dessa, trots att de är *sealed*.
- Använd arv, om de inte är *sealed* eller *static*, *extend* om de är markerade som *sealed*.
- Om klass är markerad som virtual, använd *extend*.
- Om du använder arv från abstraktklass, måste du implementera alla dess medlemmar.

Utöka .NET Framework klass

- Din klass skall lagra information linjärt och ta bort eventuella duplikat.
- Utöka List<T>-klass:

```
public class UniqueList<T> : List<T>
{
        public void RemoveDuplicates()
        {
                base.Sort();
                for (int i = this.Count - 1; i > 0; i--)
                {
                        if(this[i].Equals(this[i-1]))
                        {
                                this.RemoveAt(i);
                        }
                }
        }
}
```

.easec

Skapa skräddarsydd undantagshantering

- .NET Framework innehåller inbyggda klasser för undantagshantering, för att hantera de vanligaste felen.
- Vanligtvis används dessa klasser för att hantera undantag.
- Skapa skräddarsydd undantagshantering:
  - Arv från klassen System.Exception.
  - Implementera tre standard constructors:
    - base().
    - base(string message).
    - base(string message, Exception inner).
  - Implementera ytterligare medlemmar om behov finns.

.easec

Skräddarsydda klasser för undantag

- När du använder skräddarsydda klasser, kan du skräddarsy undantag.
- System.Exception fångar alla undantag av typen Exception.
- Börja med att fånga mer specifika undantag, därefter mer generella.
- Om skräddarsydd klass skapas för att hantera undantag, inkludera ordet Exception i slutet av namnet för din klass.

.easec

Skräddarsydda klasser för undantag - exempel

```
using System;
public class LoyaltyCardNotFoundException : Exception
{
        public LoyaltyCardNotFoundException()
        {
                // Kallar specifikt på constructor för basklass
        }
         public LoyaltyCardNotFoundException( string message) :
base(message)
        {
        }
        public LoyaltyCardNotFoundException( string message,
Exception inner) : base(message, inner)
        {
        }
}
```

.eosec

Kasta undantag

- Använd nyckelordet throw för att kasta ett undantag.

```
throw new LoyaltyCardNotFoundException()
```

- Använd try/catch block för att fånga undantaget:

```
try
{
        // Utför operation som kan leda till ett undantag
}
catch(LoyaltyCardNotFoundException ex)
{
        // Använd undantagsvariabel ex, för att få mer information
}
```

.eosec

Generic class

- När du ärver från generic class, måste du funderar på hur du vill hantera typ parametrar i basklass.
- För varje parameter måste du antingen tillhandahålla argument, eller lägga till matchande parameter till din deklaration av klass.

.eosec

Utöka funktionalitet

- I de flesta situationerna vill du utöka funktionalitet för en klass.
- Detta är dock inte möjligt om klass är sealed.
- Du kan skapa Extension Methods som ett alternativ.
- Skapa statisk metod i en statisk klass.
- Använd första parameter för att indikera typ som du vill utöka.

.easec

Utöka funktionalitet -exempel

- Exempel på att skapa utökad metod för System.String:

```
namespace FourthExtensionMethods;
{
	public static class FourthCoffeeExtensions
	{
		public static bool ContainsNumbers(this String s)
		{
		// Använder regular expression
			return Regex.IsMatch(s, @"\d");
		}
	}
}
```

- Scope:

```
using FourthExtensionMethods;
```

.easec

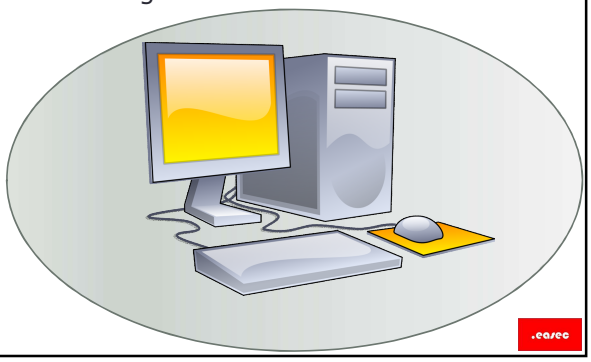
Utöka funktionalitet –exempel (forts.)

- Kalla på utökad metod:

```
Console.WriteLine("Var vänlig att skriva in text som innehåller
siffra och klicka på Enter.");
string text = Console.ReadLine();
if(text.ContainsNumbers)
{
	Console.WriteLine("Din text innehåller siffror!");
}
else
{
	Console.WriteLine("Din text innehåller inte siffror!");
}
```

.easec

Övning Arbeta med klasser och undantag



.eosec

Repetitionsfrågor

.eosec