Översikt

- Skapa och kalla på metoder.
- Arbeta vidare med metoder.
- Hantera undantag.
- Monitorera.

.eosec

Lektion 1: Skapa och kalla på metoder

- Vad är en metod?
- Skapa metod.
- Namngivning.
- Kropp.
- Specificera parameter.
- Return type.
- Kalla på metod.
- Felsöka metoder.

.eosec

Vad är en metod?

- Metoder kapslar in som skyddar information som finns lagrad.
- Applikation som du utvecklar, kommer att innehålla ett antal metoder.
- Några metoder måste finnas, t ex skrivbordsapplikation måste ha Main.
- När användare kör C#-program, kommer CLR att exekvera metoden Main i applikationen.
- .NET Framework tillhandahåller många metoder i basbibliotek för klasser.

.easec

Skapa metod

- Metod består av två delar:
  - Specifikation.
  - Kropp.
- Specifikation, består av namn för metoden, parametrar som metoden kan använda och vilken typ som kommer att returneras.
- Kropp, är ett block med programkod.
- Kombinationen namn för metoden och lista över dess parametrar bildar signatur.
- Varje metod i en klass måste ha en unik signatur.

.easec

Namngivning

- Måste starta med en bokstav eller ett understryckningstecken och kan bara bestå av:
  - Bokstäver.
  - Siffror.
  - Understryckningstecken.
- C# är skiftlägeskänslig, klass kan innehålla två metoder med samma namn, men skiljer på stora eller små bokstäver i namnet.
- Använd verb.
- Använd Pascal case, exempelvis BackColor.
- Använd Camel case, exempelvis backColor.

.easec

Kropp

- Kropp för metod består av ett block av programkod, som implementerar något av de tillgängliga programkonstruktionerna i C#.
- Omsluts med { }.
- Variabler kan deklareras inne i metoden, bara tillgängliga för metoden.

```
void StopService()
{
	var isServiceRunning = FourthCoffeService.Status;
	...
}
```

.easec

Specificera parameter

- Parametrar är lokala variabler som skapas när metoden körs och populäriseras med värden som är specificerade när metoden anropas.
- Alla metoder måste ha en lista över parametrar.
- Parametrar definieras i omgiven med paranteser efter metodens namn.
- Parameter separeras med komma om flera, om metoden inte tar någon parameter, är listan tom.
- För varje parameters specificeras typ.

.easec

Specificera parameter (forts.)

```
void StartService(int upTime, bool shutdownAutomatically)
{
	// Utför bearbetning här
}
```

- När de parametrar definieras som metoden accepterar, kan nyckelordet ref användas, detta innebär att det inte bara är värdet utan även referens

```
void StartService(ref int upTime, bool shutdownAutomatically)
{
	// Utför bearbetning här
}
```

.easec

Return Type

- Alla metoder måste ha en return type, de metoder som inte returnerar ett värde har void.
- Return type definieras innan namn på metoden.
- Om metod defineras som har return type, används return i kodblocket för din metod.

```
string GetServiceName()
{
        return ”FourthCoffee.SalesService”;
}
```

- Uttrycket som return specificera måste ha samma typ som metoden.

Kalla på metod

- Du kallar på metod, för att köra programkod i denna metod, från din applikation.
- Hur koden i metoden fungerar, behöver du inte känna till.
- För att kalla på metoden, behöver du ett namn och argument som metoden accepterar.

```
var upTime = 2000;
var shutdownAutomatically = true;
StartService(upTime, shutdownAutomatically);
void StartService(int upTime, bool shutdownAutomatically)
{
        // utför bearbetning
}
```

Kalla på metod (forts.)

- Om metod returnerar ett värde, specificerar du hur detta värde skall hanteras.
- Typiskt genom att tilldela värdet till en variabel.

```
var upTime = 2000;
string GetServiceName();
{
        return ”FourthCoffee.SalesService”;
}
```

Felsöka metoder

- Felsöking av metoden kan ske med avlusnings-funktioner i Visual Studio.
- I Visual Studio finns ett antal inbyggda funktioner för detta.
- Step Into, exekverar där du befinner dig. Om metod, kommer kod i metoden att exekveras.
- Step Over
- Step Out, exekverar kvarvarande programkod i din metod, exekvering kommer att fortsätta för återstående kod och pausas när hopp görs tillbaks till anropande kod.

.easec

Lektion 2: Arbeta vidare med metoder

- Skapa overloaded methods.
- Skapa metoder med parametertillval.
- Kalla på metod med tvingande parameter och med parametertillval.
- Kalla på metod med namngivna parametrar.
- Returnera flera värden.

.easec

Skapa overloaded methods

- Tidigare har vi hållit på med metoder som accepterar fixerat antal parametrar.
- Ibland kan du behöva en generell metod som kräver olika parametrar beroende på och hur den används.
- Detta kallas för overload methods.
- Dessa har samma namn, men måste använda sig av unik signatur.
- Signatur för metod inkluderar namn och lista över parametrar, return type finns inte med i signatur.

.easec

Skapa overloaded methods -exempel

```
void StopService();
{
        ...
}
void StopService(string serviceName)
{
        ...
}
void StopService(int serviceId)
{
}
```

- När du kallar på metoden StopService, har du ett val för vilken version du skall arbeta med, kompilator sköter resten.

.easec

Skapa metoder med parametertillval

- Ibland går det inte att använda overload, för att parametrarna inte varieras tillräckligt för att kompilator skall kunna välja metod.
- Parametertillval kan användas i detta scenario.
- Användbart också när COM-bibliotek används.
- Alla tvingande parametrar måste komma först, därefter parametertillval.

tvingande parametertillval

```
void StopService(bool forceStop, string serviceName = null, int serviceID =1)
{
        ...
}
```

.easec

Kalla på metod med tvingande parameter och med parametertillval

- Kalla på metod bara med tvingande parameter:

```
var forceStop = true;
StopService(forceStop);
```

- Kalla på metod med tvingande och parametertillval:

```
var forceStop = true;
var serviceName = ”FourthCoffee.SalesService”;
StopService(forceStop, serviceName);
```

.easec

Kalla på metod med namngivna parametrar

- Vanligtvis när du kallar på metod, motsvarar anropet parametrar i signaturen.
- Parametrar kan också specificeras med namn, detta ger möjlighet att tillhandahålla parametrar i en ***annan*** sekvens än specificerad i signatur.
- Värde separeras från parameter med kolon.

```
StopService(true, serviceID: 1);
```

.eosec

Returnera flera värden

- Metod kan skicka tillbaks ett värde, genom att använda return.
- Flera värden kan skickas tillbaks, genom att definiera parameter med prefix out.
- I kropp måste värde tilldelas parameter, innan den skickas tillbaks. Variabel måste tilldelas parameter.

.eosec

Returnera flera värden -exempel

```
bool IsServiceOnline(string serviceName, out string statusMessage)
{
        var isOnline = FourthCoffeeService.GetStatus(serviceName);
        if (isOnline)
        {
                statusMessage = ”Tjänst körs.”;
        }
        else
        {
                statusMessage = ”Tjänst stoppad.”;
        }
        return isOnline;
}
```

```
var statusMessage = string.Empty;
var isServiceOnline =
IsServiceOnline(”FourthCoffee.SalesService”. out
statusMessage);
```

.eosec

Lektion 3: Hantera undantag

- Vad är ett undantag?
- Klasser för undantag.
- Try/catch.
- Kasta ett undantag.

.easec

Vad är ett undantag?

- Att hantera undantag är viktigt.
- Din applikation skall vara designat utifrån eventuella undantag, utan att applikationen kraschar.
- Exempel:
  - Fil eller katalog inte finns.
  - Nätverksrelaterade bekymmer.
- I .NET Framework finns klasser för att hantera detta.
- När metod kastar ett undantag, måste anropande kod vara preparerat för att kunna detektera och hantera detta.

.easec

Vad är ett undantag? (forts.)

- Om anropade programkod inte tar hand om detta, kommer undantaget att propageras uppåt i koden, tills någon programkod tar ansvaret.
- Finns ingen programkod som tar ansvar, kommer exekveringen att avbrytas och meddelande kommer att visas för användaren.

.easec

Klasser för undantagshantering

- Några exempel på klasser:

Klass	Namespace	Beskrivning
Exception	System	Representerar undantag som uppstår när applikation körs.
SystemException	System	Representerar undantag som uppstår i CLR.
ApplicationException	System	Alla icke kritiska i applikationen
NullReferenceException	System	Försöker använda objekt med värde NULL.
FileNotFoundException	System.IO	Undantag om fil inte finns.
SerializationException	System.Runtime.Serialization	Undantag i process för Serialization eller Deserialization.

.easec

Try/catch

- Try/catch används för att hantera undantag på ett mer strukturerat sätt.
- Du omsluter programkod som kan gå fel, om fel uppstår skapas ett undantag, detta undantag tas om hand av din applikation.
- Finally block definieras för programkod som alltid skall köras, oavsett om undantag har uppstått eller inte.
- Om ett undantag fångas och tas om hand, kommer detta att ske innan programkod i finally körs.

.easec

Try/catch -exempel

```
try
{
}
catch (NullReferenceException) ex)
{
        // Fångar alla NullReferenceException.
}
catch (Exception ex)
{
        // Fångar alla andra undandatag.
}
finally
{
        // Programkod som alltid körs.
}
```

.easec

Kasta ett undantag

- Använd nyckelordet throw, för att kasta ett undantag.

```
var ex = new NullReferenceException(”Parameter ’Name’ är null.”);
```

- Vanlig teknik är att om inte kod för catch kan hantera undantaget, skickas undantaget tillbaks till programkod som har kallat på metoden.

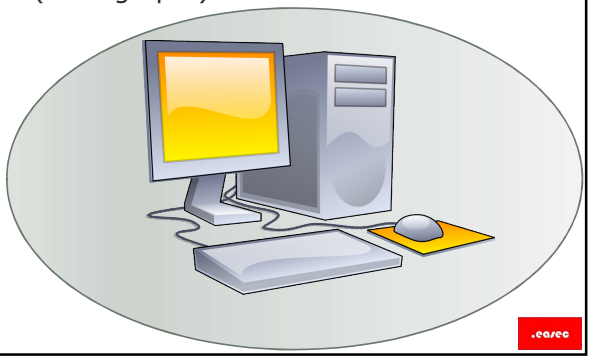
.easec

Kasta ett undantag -exempel

```
try
{
}
catch (NullReferenceException) ex)
{
        // Fångar alla NullReferenceException.
}
catch (Exception ex)
{
        // Försöker att hantera undantaget.
        ...
        // Om inte undantaget kan hanteras, kasta det till
        // anropande programkod.
        throw;
}
```

.easec

Övning Arbeta med undantag (ovning5.pdf)



.easec

Lektion 4: Monitorera

- Loggning och spårning.
- Skriva till händelselogg i Windows.
- Debug och trace.
- Application Profiling.

.eosec

Loggning och spårning

- I din applikation kan du skriva in funktioner för att lättare se vad som händer i din applikation.
- Ett sätt är att skriva till logg, antingen logg som finns i operativsystemet eller till en textfil.
- Exempelvis:
  - Om ett undantag sker, skrivs detta till händelseloggen.
- Du kan även använda verktyg i Visual Studio, oftast för att spåra eventuella värde för variabel.

.eosec

Skriva till händelselogg i Windows

- Skriva till händelselogg i Windows är ett av de vanligaste arbetsuppgifterna.
- Klassen System.Diagnostics.EventLog tillhandahåller ett antal statiska metoder som kan användas för att skriva till logg.
- EventLog.WriteEntry vanlig metod.
- För att skriva till händelselogg i Windows krävs tre saker:
  - Namn på eventlogg.
  - Källa för event.
  - Meddelande.

.eosec

Skriva till händelselogg i Windows (forts.)

- Skriva till händelselogg kräver administratörsrättigheter, undantag kommer att kastas, SecurityException, om applikation inte körs med högre rättigheter.

```
string eventLog = ”Application”;
string eventSource = ”Demonstration”;
string eventMessage = ”Meddelande från applikation”;
// Skapa källa för händelse om det inte redan finns
if (!EventLog.Source.Exists(eventSource))
      EventLog.CreateEventSource(eventSource, eventlog);
// Logga meddelandet
EventLog.WriteEntry(eventSource, eventMessage);
```

.easec

Debug och trace

- Namespace System.Diagnostics innehåller två klasser: debug och trace.
- Dessa två klasser kan användas för att monitorera exekveringen av din applikation.
- Fungerar på ett likvärdigt sätt och innehåller många liknande metoder.
- Skillnad mellan dessa är att debug är endast tillgängligt när du bygger din lösning i läget Debug.

.easec

Debug och trace (forts.)

- Debug och trace innehåller metoder för att skriva till Outputfönstret, eller till annan lyssnare.
- Bägge innehåller metod med namnet Assert.
- Assert ger möjlighet att specificera ett villkor (antingen true eller false) tillsammans med sträng.
- Om villkoret undersöks till false, kommer exekveringen att avbrytas och dialogruta kommer att visas med meddelandet som du specificerade.

.easec

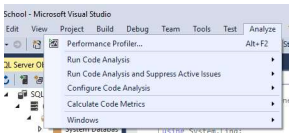
Debug och trace -exempel

```
int number;
Console.WriteLine(”Skriv in ett nummer mellan 1 - 10, klicka
Enter”;
string userInput = Console.ReadLine();
Debug.Assert(int.TryParse(userInput, out number),
   string.Format(”Kunde inte läsa {0} som integer”, userInput);
Debug.WriteLine(Nuvarande värde för userInput är: {0}”, userInput);
Debug.WriteLine(Nuvarande värde för nummer är: {0}”, number);
Console.WriteLine(”Klicka på tangent för att avsluta”);
Console.ReadLine();
```

.eosec

Application Profiling

- Programkod utan buggar är ett arbete, tänk också på att försäkra dig om bra prestanda.
- I Visual Studio finnd ett antal verktyg för detta arbete.
- Skapa och kör en Performance session:



.eosec

Repetitionsfrågor

.eosec