Översikt

- Implementera Multitasking.
- Utföra operationer asynkront.
- Synkronisera access till informationen.

.easec

Lektion 1: Implementera Multitasking

- Skapa task.
- Klassen Task.
- Starta Task.
- Task.Start, TaskFactory.StartNew.
- Task.Run.
- Vänta på Task.
- Returnera värde från task.
- Avbryta task som körs länge.
- Köra tasks parallellt.
- Länka tasks.
- Hantera undantag.

.easec

Skapa task

- Typisk består grafisk applikation av kod som körs när händelse uppstår.
- Dessa händelser svarar på händelser som musklick, flyttar på musen eller öppnar ett fönster.
- Som standard körs koden i UI thread, tänk på att inte köra långa sekvenser i denna tråd, kan innebär att det grafiska gränssnitt slutar att svara.
- Att köra allt i en tråd, nyttjar inte kapaciteten i våra dator till fullo.

Skapa task (forts.)

- En tråd – en processor core.
- .NET Framework inkluderar Task Parallel Library, ett antal klasser som gör det enkelt att distribuerar exekvering över flera trådar eller flera processorer.
- Arbetsuppgifter som tar längre tid, kan tilldelas separat thread, lämnar UI thread ledigt för respons från användare.

Klassen Task

- Klassen Task är själva hjärtat i Task Parallell Library.
- Klassen används för att representerar arbetsuppgift och gör det möjligt att utföra flera uppgifter samtidigt, i skilda trådar.
- Task Parallell Library hanterar thread pool och tilldelar arbetsuppgifter till tråd.
- Task object kör block av kod, kod specificeras som parameter till constructor.

Klassen Task (forts.)

- Du kan tillhandahålla kod i en metod och skapa Action delegate för att kapsla in metoden.

```
Task task1 = new Task(new Action(MyMethod));
```

- Alternativt anonymous delegate/anonymous method.

```
Task task2 = new Task(delegate
{
        Console.WriteLine(”Task 2 reporting”);
});
```

.easec

Klassen Task (forts.)

- Använda lambda expression (rekommenderas).

```
Task task2 = new Task(() =>
{
        Console.WriteLine(”Task 2 reporting”);
});
```

- => goes to.

.easec

Kontrollera exekvering av Task

- Task Parallel Library erbjuder ett antal olika metoder att starta tasks, finns också olika sätt att pausa exekvering av kod, tills en eller flera tasks har körts klart.
- När programkod startar task, kommer Task Parallel Library tilldela thread till task och även att köra den.
- Task och program körs i olika threads.

.easec

Starta task

- Att starta task:
  - Task.Start (instans metod).
  - Task.Factory.StartNew (statisk metod).
  - Task.Run (statisk metod).
- Att vänta på att task skall slutföras:
  - Task.Wait (instans metod).
  - Task.WaitAll (statisk metod).
  - Task.WaitAny (statisk metod).

.easec

Task.Start, TaskFactory.StartNew

- Att använda Task.Start:

```
var task1 = new Task( () => Console.WriteLine("Task 1 har körts
klart.") );
task1.Start();
```

- Att använda TaskFactory.StartNew:

```
var task3 = Task.Factory.StartNew( () => Console.WriteLine("Task 3
har körts klart.") );
```

.easec

Task.Run

- Att använda Task.Run:

```
var task4 = Task.Run( () => Console.WriteLine("Task 4 har körts
klart.") );
```

.easec

Vänta på task

- Ibland vill du pausa exekvering av din programkod tills task har exekverat klar.
- Klassen erbjuder ett antal metoder för detta:
  - Om du vill vänta på en task skall slutföras, använd metoden Task.Wait.
  - Om du vill vänta på att flera task skall slutföras, använd statisk metod Task.WaitAll.
  - Om du vill vänta på att någon task i en samling av task skall slutföras, använd metoden Task.WaitAny.
- Task.WaitAll och Task.WaitAny, läggs tasks i en array.

.easec

Returnera värde från task

- För att task skall bli effektiva, så måste task kunna returnera värde, ett resultat.
- Den vanliga Task klassen ger oss inte denna möjlighet, men Task Parallel Library tillhandahåller Task<TResult>, som är en klass av typen generic.
- När du skapar en instans av Task<TResult>, används type parameter för att specificera typ av resultat som task returnerar.
- Read-only property Result, används för att hämta värde för att returnera till din programkod.

.easec

Returnera värde från task -exempel

```
// Skapa och köar task som returnerar veckodag som en sträng.
Task<string> task1 = Task.Run<string>( () =>
DateTime.Now.DayOfWeek.ToString() );
Console.WriteLine(task1.Result)
// Om du försöker få tillgång till Result, innan task har slutförs,
kommer programkod att vänta tills ett resultat finns att tillgå.
```

.easec

Avbryta task som körs länge

- Tasks används oftast för att utföra operationer som tar lång tid, utan att blockera UI thread.
- Ibland vill du ge möjlighet för användare att avbryta detta arbete.
- I Task Parallell Library används cancellation tokens, för att ge stöd för att kunna avbryta task utan att det får konsekvenser.

Avbryta task som körs länge -steg

- Skicka med cancellation token som ett argument till delegate method.
- I thread som har skapad task, begär cancellation genom att kalla på metoden Cancel.
- Kontrollera om cancellation token är avbruten.
- Returnera eller kasta (throw) undantaget OperationCanceledException.

Köra tasks parallellt

- Task Parallel Library inkluderar statisk klass med namnet Parallel.
- Klassen tillhandahåller ett antal metoder som du kan använda för att köra tasks simultant.
- Om du vill köra ett antal specificerade task parallellt, använder du metoden *Parallel.Invoke*.
- *Parallel.For* för att köra for-loop parallellt.
- *Parallel.ForEach* för att köra foreach-loop parallellt.
- *PLINQ* kan användas för att köra LINQ-uttryck parallellt.

Länka tasks

- Task.ContinueWith metoden används för att länka ihop tasks.
- Nestade task kan användas för att starta oberoende task.
- Child task kan användas för att skapa task med beroende.

NestedTasks_5

.easec

Hantera undantag

- Kalla på Task.Wait för att fånga propagerade undantag.
- Fånga AggregateException i ett catch block.
- Hantera property InnerException för att hantera individuella undantag.

TaskExceptions

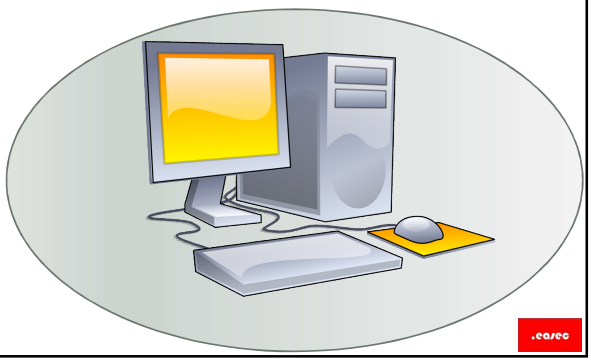
.easec

Hantera undantag -exempel

```
try
{
        task1.Wait();
}
catch(AggregateException ae)
{
        foreach(var inner in ae.InnerExceptions)
        {
                // Ta hand om undantaget
        }
}
```

.easec

Övning Filer Modul 12 Lektion 1



Lektion 2: Utföra operationer asynkront

- Introduktion till asynkrona operationer.
- Använda Dispatcher.
- Använda async och await.
- Skapa metoder som väntar.
- Skapa CallBack metod.
- Arbeta med APM.
- Hantera undantag asynkront.

Introduktion till asynkrona operationer

- Asynkrona operationer är operationer som körs i separata threads, thread som initierar en asynkron operation behöver inte vänta på att operationen skall slutföras, för att fortsätta.
- Asynkrona operationer är tät relaterade till tasks.
- Nyckelorden async och await används för att köra asynkrona operationer.

Använda Dispatcher

- I .NET Framework är varje thread associerad med ett Dispatcher objekt.
- Dispatcher är ansvarig för att underhålla kö med arbetsuppgifter för thread.
- Dispatcher kan användas i grafisk applikation, för att skicka uppdateringar från bakgrunden till UI.
- Dispatcher.BeginInvoke.

```
lblTime.Dispatcher.BeginInvoke(new Action(() =>
    SetTime(currentTime)));
```

Dispatcher

.eo/ee

Använda async och await

- Lägg till *async* modifier till deklaration för metod.
- Använd *await* inuti async metod för att vänta på att task skall slutföras, utan att blockera thread.

```
private async void btnLongOperation_Click(object sender, RoutedEventArgs e)
{
   ...
   Task<string> task1 = Task.Run<string>(() =>
      {
         ...
      }
  lblResult.Content = await task1;
}
```

GetCoffee

.eo/ee

Skapa metoder som väntar

- Operator await används alltid för att vänta på att task skall slutföras.
- Om din synkrona metod returnerar void, asynkrona motsvarigheten kommer att returnera Task.
- Om din synkrona metod returnerar T som typ, asynkrona motsvarigheten kommer att vara Task<T>.

.eo/ee

Skapa CallBack metod

- När asynkron metod är klar, kan denna metod kalla på callback method.
- Den asynkrona metoden skickar informationen till callback method, där informationen kommer att bearbetas.
- Metoden används också när ett UI skall uppdateras.

.easec

Arbeta med APM

- Många .NET Framework klasser implementerar asynkrona operationer efter designmönstret APM.
- Detta designmönster implementeras som två metoder, BeginNamn_på_operation och EndNamn_på_operation.
- Klassen HttpWebRequest inkluderar metoder med namnen BeginGetResponse och EndGetResponse.

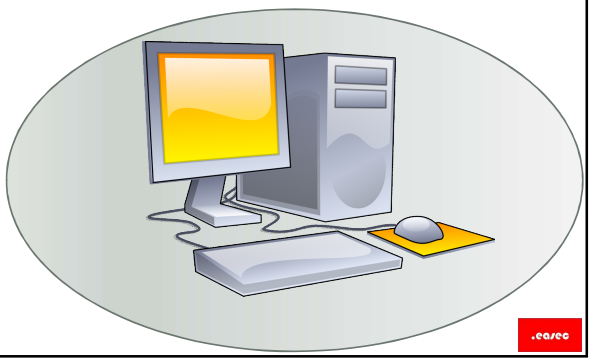
.easec

Hantera undantag asynkront

- När nyckelorden async och await används, kan undantag hanteras på samma sätt som de hanteras synkront, dvs med try/catch.
- Trots att metoder är asynkrona, kommer undantag att rapporteras till program som har kallat på metoden. Bakom detta ligger Task Parallel Library.

.easec

Övning Filer Modul 12 Lektion 2



.easec

Lektion 3: Synkronisera access till informationen

- Använda Locks.
- Synchronization Primitives.
- Concurrent Collections.

.easec

Använda Locks

- När multithreading används i din applikation, så kan situationer uppstå där flera threads har tillgång till din resurs, exempelvis för att hålla reda på lagerhållningen.
- Nyckelordet lock används för att ”stänga ute” andra trådar.

.easec

Synchronization Primitives

- Synchronization Primitives är mekanismer som tillåter dess användare, i detta fallet .NET runtime, att kontollera synkroniseringen av trådar.
- Task Parallel Library stödjer ett stort antal sådana Primitives, som kontrollerar tillgång till resurser på olika sätt.

.easec

Synchronization Primitives (forts.)

- *Klass ManualResetEventSlim*, används för att begränsa tillgång till resurs till ett thread åt gången.
- *Klass SemaphoreSlim*, begränsar tillgången till resurser till ett fixerat antal threads.
- *Klass CountDownEvent*, begränsar tillgång till resurs, till dess att ett satt värde av tasks har signalerats att vara klara.
- *Klass ReadWriterLockSlim*, ger möjlighet till flera threads att läsa resurs, när som helst.

.easec

Synchronization Primitives (forts.)

- *Klass Barrier*, används för att blockera flera threads, tills dess att alla uppfyller ett givet villkor.

.easec

Concurrent Collections

- Standardklasser för samlingar i .NET Framework, är som standard, inte thread-safe.
- När multithreading används är det viktigt att du inte påverkar intigritet för samlingen.
- En teknik att använda är synchronization primitives, en annan är att använda samlingar som är specifikt designade för multithreading.
- Finns i System.Collection.Concurrent.

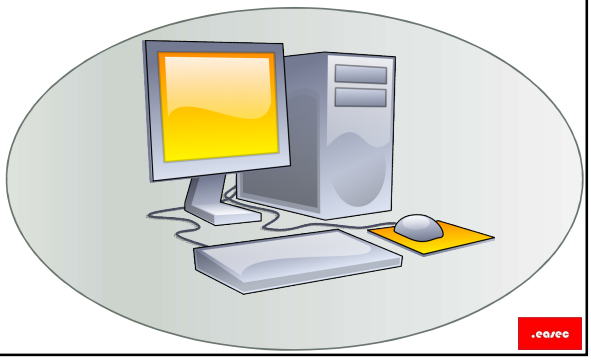
.easec

Concurrent Collections (forts.)

Klass eller interface	Beskrivning
ConcurrentBab<T>	Klass tillhandahåller ett säkert sätt att lagra oordnad samling av enheter.
ConcurrentDictionary<T>	Klass tillhandahåller ett thread-safe alternativ till Dictionary<TNyckel, Tvärde>-klassen.
ConcurrentQueue<T>	Klass tillhandahåller ett thread-safe alternativ till Stack<T>-klassen.
IProducerConsumerCollection<T>	Interfacet definierar metoder för att implementera klasser som exponerar beteende som producent/konsumerare. Producent lägger till, konsument läser från samling. Klasserna ovan använder interfacet.
BlockingCollection<T>	Klassen fungerar som wrapper för samlingar som implementerar IProducerConsumerCollection<T>. Används exempelvis för att förhindra ett tillägg till samlingen tills utrymme finns tillgängligt.

.easec

Övning Använda Lock



.easec

Repetitionsfrågor