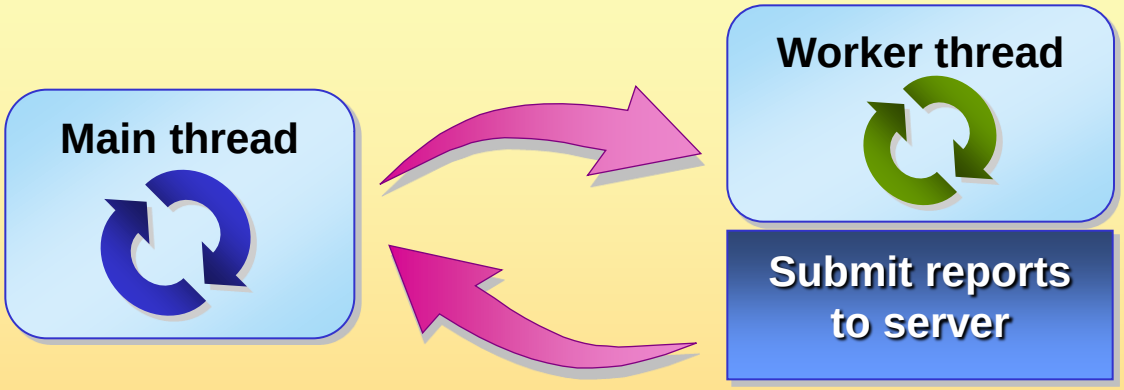
Asinhrono programiranje

Programiranje korisničkih interfejsa

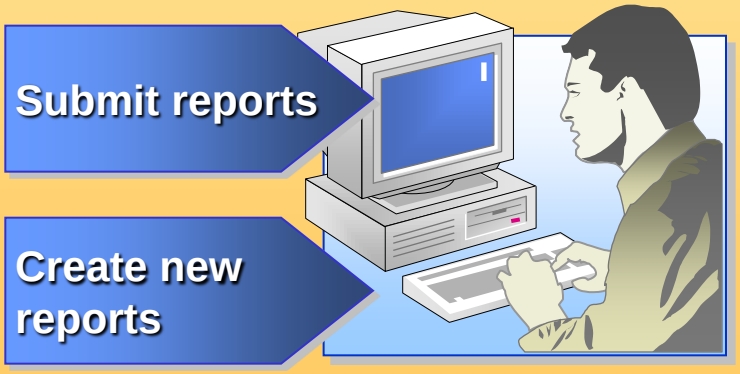
Bojan Furlan

What Is Asynchronous Programming?

- An application gives some work to other thread(s) while it continues doing other work on the main thread



- Useful in Windows Forms applications so users will not be blocked waiting for results and can instead continue with other work



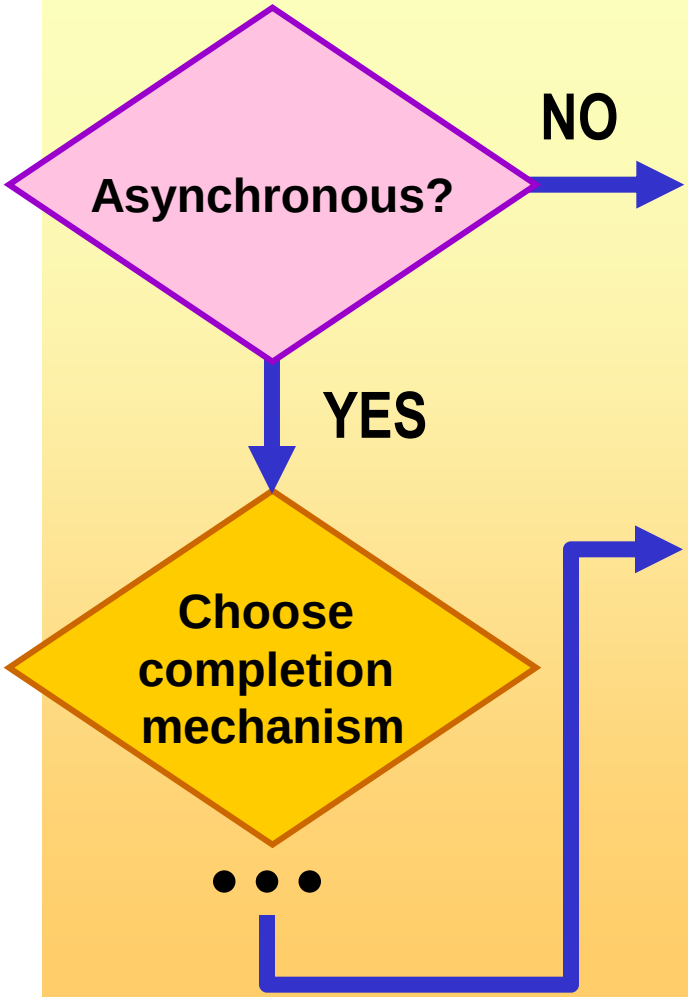
Demonstration: Comparing Synchronous and Asynchronous Versions of an Application

In this demonstration, you will be able to compare the user experience in synchronous and asynchronous versions of the application

Asynchronous Programming Support in the .NET Framework

- **A design pattern for asynchronous programming**
  - Used by the .NET Framework to make asynchronous calls uniform across different parts of the framework
  - User-created classes that support asynchronous calls should conform to this design pattern
- **Asynchronous support is provided in many of the logical areas**
  - I/O, sockets, networking, ASP.NET and XML Web services, messaging, and asynchronous delegates
  - Implementation is transparent, call the appropriate methods and let the NET Framework handle the details

The Asynchronous Programming Model Design Pattern




```
// ...
// ... call Operation
// ... wait for return
// ... continue processing
// ...
```

```
// ...
// ... call BeginOperation
// ... operation begun on
//     another thread
// ... continue with other
//     processing
// ... receive results
// ... process results
// ...
```

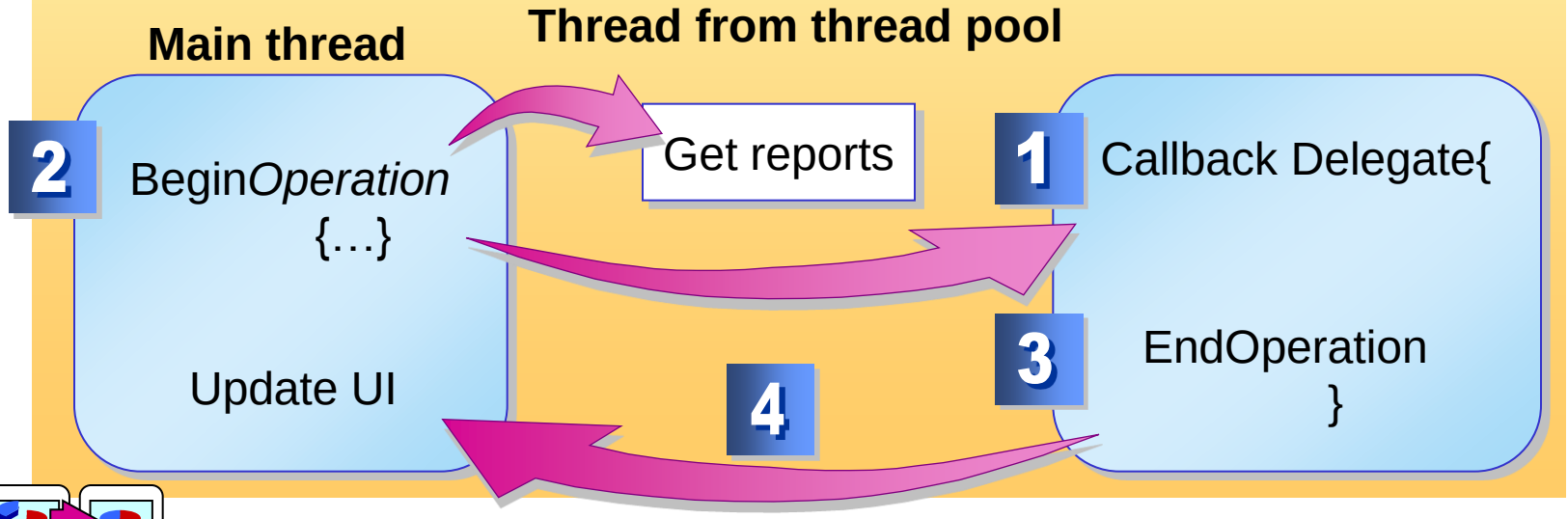
Overview of the Asynchronous Programming Model Design Pattern

- **Caller decides whether a particular call should be asynchronous**
- **Asynchronous operation logically split into two parts**
  1. Client begins the operation by calling the Begin*Operation* method
  2. Client notified that operation is complete and receives results

Completion Technique	Comments
Use a callback	Supply a callback delegate, method will be called when operation completes (no blocking)
Poll	Poll the **IAsyncResult** interface's **IsCompleted** property
Call the End*Operation* method	Call the End*Operation* method and block till operation completes (Problem: infinite blocking)
Wait on a handle	Wait on **IAsyncResult** interface's **WaitHandle** property, then call End*Operation* method

Using the Design Pattern with an Asynchronous Callback for Completion

1. Create the asynchronous callback delegate
2. Invoke the Begin*Operation* method, passing it the callback delegate
3. Inside the callback, invoke the End*Operation* method to notify that asynchronous work is complete and to return results
4. Return control to the main thread and update UI



How to Set Up and Initiate the Call

1 Create the asynchronous callback delegate

Asynchronous callback delegate

```
AsyncCallback delCB = new AsyncCallback(
        this.AsyncCB);
```

2 Invoke the Begin*Operation* method, passing it the callback delegate

Invoke the Begin*Operation* method

```
WS.BeginGetReportsForEmployee(
        username, pwdToken,
        RecordCursor, 10, TotalNumRecords,
        delCB, null);
```

Callback delegate is passed in to the Begin*Operation* method

How to Receive Completion Notification and Results

3 **Inside the callback, invoke the End*Operation* method to retrieve the results of the asynchronous call**

```
// Inside the callback method, AsyncCB, call
// End Operation to get results of the async call

void AsyncCB (IAsyncResult ar)
{
                    ...
DataSet ds = WS.EndGetReportsForEmployee(
             ar, out TotalNumRecords);
                    ...
}
```

Invoke End*Operation* method

Receive results

How to Return Control to the Main Thread

In Windows Forms applications, any calls to methods or properties for controls on the form must be done on the main thread

4 Return control to the main thread

```
//Switch back to main thread to update the UI
//First, create a MethodInvoker delegate for
//the method to be called
MethodInvoker mi = new MethodInvoker(
    this.UpdateUI);

// Use the current form's BeginInvoke to
// invoke the delegate
this.BeginInvoke(mi);
```

Overview of How to Make Asynchronous Calls to Any Existing Method

1. **You must explicitly create and call a delegate for the method that you want to invoke**

2. **Follow the design pattern for asynchronous programming**

   - Initiate the call
   - Complete the call
   - Return data (if applicable) and control to the main thread

How to Create the Asynchronous Delegate

1 Declare the delegate

Delegate keyword

```
public delegate int CalcDelegate(
   int startingValue,
   int interestRate);
```

The delegate's signature matches that of the method it will point to

2 Instantiate the delegate, passing in the method that the delegate points to

```
//Instantiate class that contains method delegate points to
TotalReturnCalc tr = new TotalReturnCalc();
//Instantiate the delegate, passing it the method to call
CalcDelegate cd = new CalcDelegate(tr.CalculateReturn);
```

The method that you want the delegate to point to

How to Initiate the Asynchronous Call

1. **Create the delegate to the callback method**

2. **Call the BeginInvoke method**
   - When using a callback method, pass in the delegate for the callback method
   - Returns an object implementing **IAsyncResult**

Method that will receive the callback notification

```
// create AsyncCB delegate to callback method
AsyncCallback cb = new AsyncCallback(this.ResultsCB);

// call BeginInvoke to asynchronously call the method
IAsyncResult ar = cd.BeginInvoke(startVal, intRate, cb, null);
```

Callback delegate passed in to BeginInvoke

How to Complete the Asynchronous Call

1 Call the EndInvoke method

- Returns a return value or a data structure that includes a return value

```
//inside the callback method called ResultsCB
void ResultsCB(IAsyncResult ar)
{
   ...
   int result = cd.EndInvoke(ar);
   ...
}
```

Use EndInvoke to return results

2 Update the UI to reflect the results of the operation

When using Windows Forms, this involves returning control back to the main UI thread because Windows Forms can only be safely called from the main thread

How to Return Control to the Main Thread and Update the UI

1 **Instantiate a MethodInvoker delegate for the UI update method**

```
//Switch back to main thread before updating UI

MethodInvoker mi = new MethodInvoker(this.UpdateUI);
```

2 **Asynchronously call the MethodInvoker delegate**

```
// Use BeginInvoke to call the MethodInvoker

this.BeginInvoke(mi);
```

Practice: Making an Asynchronous Call

In this practice, you will

- Modify the application so that it makes asynchronous calls
- Rebuild the application and observe how the behavior of the application has changed

Begin reviewing the objectives for this practice activity

How to Protect State and Data in a Multithreaded Environment

- **Synchronized code region**

  Monitor class
- **Manual synchronization**
  - **Mutex** class
  - **ReaderWriterLock** class
  - **Interlocked.Increment** and **Interlocked.Decrement** methods
- **Design applications to try to minimize synchronization needs**

Reference

- **Threading in .NET and WinForms**
  - http://www.codeproject.com/KB/threads/Threading.aspx