TDDD49
C# and .NET Programming

(Lecture 03)

Sahand Sadjadee
Department of Information
and Computer Science
Linköping University

Outline

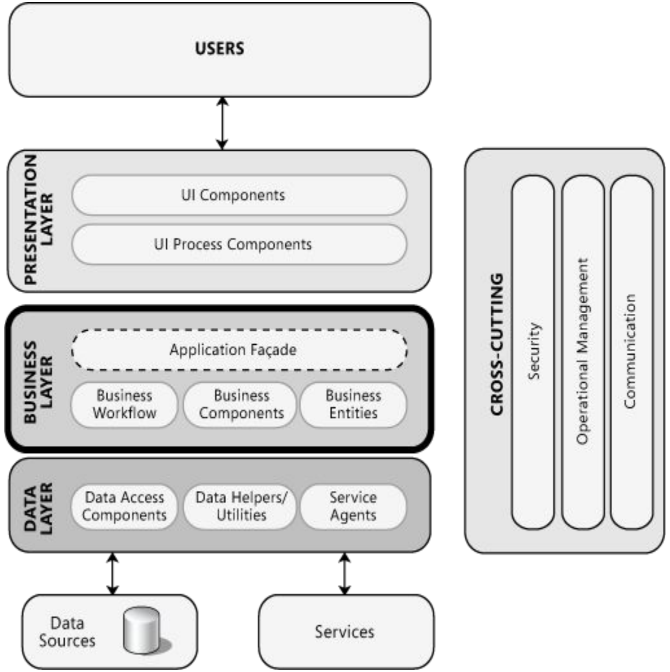
1. The Business Logic Layer(BLL)
2. Multithreading
3. Networking

LINKÖPINGS UNIVERSITET

The Business Logic Layer

The Business Logic Layer

https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ee658103(v%3dpandp.10)



- **Application façade.** This optional component typically provides a simplified interface to the business logic components, often by combining multiple business operations into a single operation that makes it easier to use the business logic.
- **Business Workflow components**. After the UI components collect the required data from the user and pass it to the business layer, the application can use this data to perform a business process. Many business processes involve multiple steps that must be performed in the correct order, and may interact with each other through an orchestration.
- **Business Entity components.** Business entities, or—more generally—business objects, encapsulate the business logic and data necessary to represent real world elements, such as Customers or Orders, within your application.

LINKÖPINGS UNIVERSITET

General Design Considerations

- **Decide if you need a separate business layer.** It is always a good idea to use a separate business layer where possible to improve the maintainability of your application. The exception may be applications that have few or no business rules (other than data validation).
- **Identify the responsibilities and consumers of your business layer.** This will help you to decide what tasks the business layer must accomplish, and how you will expose your business layer. Use a business layer for processing complex business rules, transforming data, applying policies, and for validation.
- **Do not mix different types of components in your business layer.** Use a business layer to avoid mixing presentation and data access code with business logic code, to decouple business logic from presentation and data access logic, and to simplify testing of business functionality.
- **Reduce round trips when accessing a remote business layer.**
- **Avoid tight coupling between layers.** Use the principles of abstraction to minimize coupling when creating an interface for the business layer.

LINKÖPINGS UNIVERSITET

Relevant Design Patterns

- **Anemic Model**
  - Centralize and aggregate behavior to provide a uniform service layer.
  - Entities do not contain logic.
  - There are services/classes which contain the whole logic.
  - Criticism: “The fundamental horror of this anti-pattern is that it's so contrary to the basic idea of object-oriented designing; which is to combine data and process them together. The anemic domain model is just a procedural style design, exactly the kind of thing that object bigots like me ... have been fighting since our early days in Smalltalk. What's worse, many people think that anemic objects are real objects, and thus completely miss the point of what object-oriented design is all about.” https://martinfowler.com/bliki/AnemicDomainModel.html
- **Domain Model**
  - A set of business objects that represents the entities in a domain and the relationships between them.
  - Entities contain both logic and data.

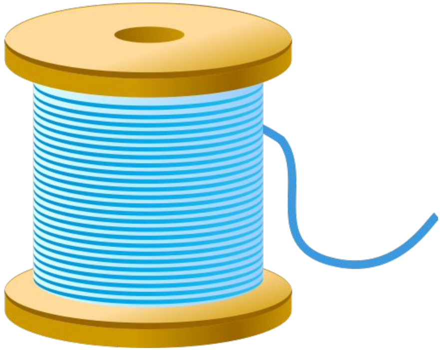
LINKÖPINGS UNIVERSITET

Project Structure

Business-Logic code (→ Model)

..	
HotelDbContext	before single responsibility
Miscellaneous	Hotel rooms and client work correctly.
Model	Hotel rooms and client work correctly.
Properties	changed database server
View	Exporting client information to xls
ViewModel	Exporting client information to xls
App.config	some ado.net examples for lab 8
App.xaml	changed database server
App.xaml.cs	changed database server
HotelSystem.csproj	Exporting client information to xls
ReportCreator.cs	Exporting client information to xls
packages.config	changed database server

LINKÖPINGS UNIVERSITET



Multithreading

When to use multiple threads

- You use multiple threads to increase the responsiveness of your application.
- The key is to not use the main thread for performing time-consuming tasks.
- Dedicated threads can be used for network and device communication to be more responsive to incoming messages.

How To Create threads

https://msdn.microsoft.com/en-us/library/btky721f.aspx

1. Declare the thread.
   - `System.Threading.Thread myThread;`
2. Create an instance of the thread with the appropriate delegate for the starting point of the thread.
   - `myThread = new System.Threading.Thread(new System.Threading.ThreadStart(myStartingMethod));`
3. When ready, call the Thread.Start method to start the thread.
   - `myThread.Start();`

System.Threading.Thread is a foreground thread.

LINKÖPINGS UNIVERSITET

Background vs foreground threads

- Background threads can't prevent the application from terminating while foreground threads continue execution even the main thread has terminated.
- Once all foreground threads have been stopped in a managed process (where the .exe file is a managed assembly), the system stops all background threads and shuts down.
- When the runtime stops a background thread because the process is shutting down, no exception is thrown in the thread.
- an unhandled exception in either foreground or background threads results in termination of the application.
- Threads that belong to the managed thread pool (that is, threads whose IsThreadPoolThread property is true) are background threads.
- All threads that enter the managed execution environment from unmanaged code are marked as background threads.
- All threads generated by creating and starting a new Thread object are by default foreground threads.

LINKÖPINGS UNIVERSITET

ThreadPool

https://docs.microsoft.com/en-us/dotnet/api/system.threading.threadpool?view=netframework-4.7.2

Provides a pool of background threads that can be used to:

- execute tasks
- post work items
- process asynchronous I/O
- wait on behalf of other threads
- process timers.

LINKÖPINGS UNIVERSITET

ThreadPool

https://docs.microsoft.com/en-us/dotnet/api/system.threading.threadpool?view=netframework-4.7.2

- Many applications create threads that spend a great deal of time in the sleeping state, waiting for an event to occur. Other threads might enter a sleeping state only to be awakened periodically to poll for a change or update status information.
- The thread pool enables you to use threads more efficiently by providing your application with a pool of worker threads that are managed by the system.

ThreadPool

https://docs.microsoft.com/en-us/dotnet/api/system.threading.threadpool?view=netframework-4.7.2

Where thread pool is used:

- When you create a Task or Task<TResult> object to perform some task asynchronously, by default the task is scheduled to run on a thread pool thread.
- Asynchronous timers use the thread pool. Thread pool threads execute callbacks from the System.Threading.Timer class and raise events from the System.Timers.Timer class.
- When you use registered wait handles, a system thread monitors the status of the wait handles. When a wait operation completes, a worker thread from the thread pool executes the corresponding callback function.
- When you call the QueueUserWorkItem method to queue a method for execution on a thread pool thread.

LINKÖPINGS UNIVERSITET

ThreadPool

https://docs.microsoft.com/en-us/dotnet/api/system.threading.threadpool?view=netframework-4.7.2

```
    public static void Main()
    {
        // Queue the task.
        ThreadPool.QueueUserWorkItem(ThreadProc);
        Console.WriteLine("Main thread does some work, then sleeps.");
        Thread.Sleep(1000);

        Console.WriteLine("Main thread exits.");
    }

    // This thread procedure performs the task.
    static void ThreadProc(Object stateInfo)
    {
        // No state object was passed to QueueUserWorkItem, so stateInfo is null.
        Console.WriteLine("Hello from the thread pool.");
    }
```

LINKÖPINGS UNIVERSITET

Task-based Asynchronous Pattern (TAP)

https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap?view=netframework-4.7.2

The Task-based Asynchronous Pattern (TAP) is based on the System.Threading.Tasks.Task and System.Threading.Tasks.Task<TResult> types in the System.Threading.Tasks namespace, which are used to represent arbitrary asynchronous operations.

LINKÖPINGS UNIVERSITET

Tasks

- The Task class represents a single operation that does not return a value and that usually executes asynchronously.
- Task objects are one of the central components of the task-based asynchronous pattern first introduced in the .NET Framework 4.
- You can use the Status property, as well as the IsCanceled, IsCompleted, and IsFaulted properties, to determine the state of a task.
- A lambda expression is used to specify the work that the task is to perform.

LINKÖPINGS UNIVERSITET

Tasks

```
Action<object> action = (object obj) =>
                                {
                                   Console.WriteLine("Task={0}, obj={1}, Thread={2}",
                                   Task.CurrentId, obj,
                                   Thread.CurrentThread.ManagedThreadId);
                                };

        // Create a task but do not start it.
        Task t1 = new Task(action, "alpha");

        // Construct a started task
        Task t2 = Task.Factory.StartNew(action, "beta");
        // Block the main thread to demonstrate that t2 is executing
        t2.Wait();

        // Launch t1
        t1.Start();
        Console.WriteLine("t1 has been launched. (Main Thread={0})",
                          Thread.CurrentThread.ManagedThreadId);
        // Wait for the task to finish.
        t1.Wait();
```

LINKÖPINGS UNIVERSITET

Task<TResult> Class

https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task-1?view=netframework-4.7.2

- The Task<TResult> class represents a single operation that returns a value and that usually executes asynchronously.

Task<TResult> Class

https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.task-1?view=netframework-4.7.2

```
var t = Task<int>.Run( () => {
                                      // Just loop.
                                      int max = 1000000;
                                      int ctr = 0;
                                      for (ctr = 0; ctr <= max; ctr++) {
                                         if (ctr == max / 2 && DateTime.Now.Hour <= 12) {
                                            ctr++;
                                            break;
                                         }
                                      }
                                      return ctr;
                                    } );
      Console.WriteLine("Finished {0:N0} iterations.", t.Result);
```

LINKÖPINGS UNIVERSITET

System.Threading.Timer Class

https://docs.microsoft.com/en-us/dotnet/api/system.threading.timer?view=netframework-4.7.2

- Provides a mechanism for executing a method on a thread pool thread at specified intervals.
- This class cannot be inherited.

System.Timers.Timer Class

https://docs.microsoft.com/en-us/dotnet/api/system.timers.timer?view=netframework-4.7.2

- Generates an event after a set interval, with an option to generate recurring events.

BackgroundWorker Class https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.backgroundworker?view=netframework-4.7.2

- Executes an operation on a separate background thread.
- BackgroundWorker is used for keeping the UI responsive.
- Can be used for doing operations like downloading and database transactions.

ThreadPriority Enum

https://docs.microsoft.com/en-us/dotnet/api/system.threading.threadpriority?view=netframework-4.7.2

- Specifies the scheduling priority of a Thread.
- Thread priorities specify the relative priority of one thread versus another.

Threading model in WPF

https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/threading-model

- WPF applications start with two threads: one for handling rendering and another for managing the UI.
- The UI thread receives input, handles events, paints the screen, and runs application code.
- The UI thread queues work items inside an object called a Dispatcher which selects work items on a priority basis and runs each one to completion.
- It's recommended to minimize the tasks' size in order to increase the Dispatcher throughput and UI responsive.
- For big operations you need to use separate threads which report the result to the UI thread upon completion.
- **Other threads than UI thread do not have the right to update the UI components directly.**

LINKÖPINGS UNIVERSITET

WPF Dispatcher

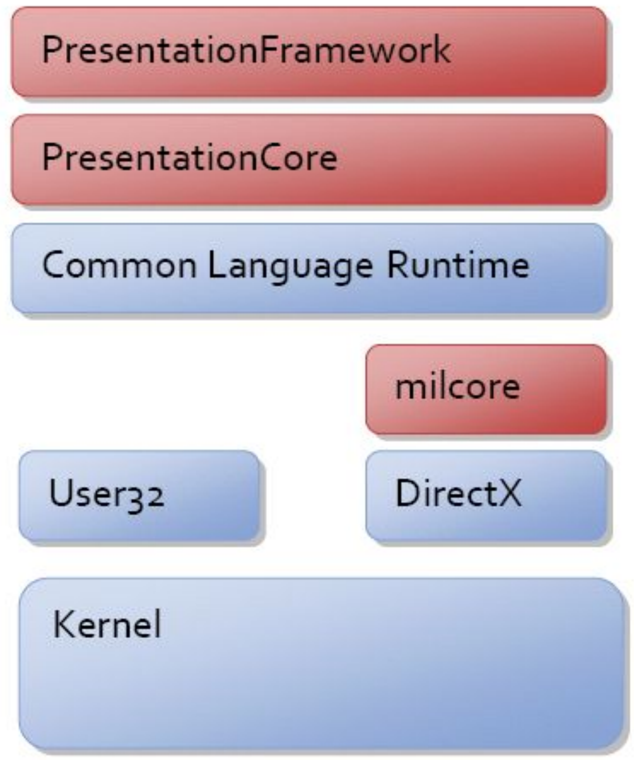
https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/threading-model

- Dispatcher is a queue which is associated with the UI thread.
- Dispatcher queues method calls.
- Only Dispatcher can update the objects in the UI from non-UI thread.
- DispatcherObject allows access to the dispatcher.
- **Most classes in WPF derive from DispatcherObject.**
- At construction, a DispatcherObject stores a reference to the Dispatcher linked to the currently running thread.

LINKÖPINGS UNIVERSITET

WPF Architecture

https://msdn.microsoft.com/en-us/library/ms750441(v=vs.110).aspx



Key classes:

System.Threading.DispatcherObject

System.Windows.DependencyObject

System.Windows.Media.Visual

System.Windows.UIElement

System.Windows.FrameworkElement

System.Windows.Controls.Control

DispatcherObject

https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/threading-model

- Dispatcher is a queue which is associated with the UI thread.
- Dispatcher queues method calls.
- Only Dispatcher can update the objects in the UI from non-UI thread.
- DispatcherObject allows access to the dispatcher.
- **Most classes in WPF derive from DispatcherObject.**
- At construction, a DispatcherObject stores a reference to the Dispatcher linked to the currently running thread.

DispatcherObject

https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/threading-model

- CheckAccess() Determines whether the calling thread has access to this DispatcherObject.
- VerifyAccess() Enforces that the calling thread has access to this DispatcherObject.
- Dispatcher Gets the Dispatcher this DispatcherObject is associated with.

Dispatcher

https://docs.microsoft.com/en-us/dotnet/api/system.windows.threading.dispatcher?view=netframework-4.7.2

BeginInvoke - executes code asynchronously

Invoke - executes code synchronously

Managed threading best practices https://docs.microsoft.com/en-us/dotnet/standard/threading/managed-threading-best-practices

- You use multiple threads to increase the responsiveness of your application.
- The key is not use the main thread for performing time-consuming tasks.
- Dedicated threads can be used for network and device communication to be more responsive to incoming messages.

Networking

System.Net.Sockets

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/sockets

- The System.Net.Sockets namespace contains a managed implementation of the Windows Sockets interface.
- All other network-access classes in the System.Net namespace are built on top of this implementation of sockets.
- The Socket class supports two basic modes, synchronous and asynchronous.

LINKÖPINGS UNIVERSITET

How to: Create a Socket

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/how-to-create-a-socket

- **Using TCP**
  - ```
    Socket s = new Socket(AddressFamily.InterNetwork,
       SocketType.Stream, ProtocolType.Tcp);
    ```
- **Using UDP**
  - ```
    Socket s = new Socket(AddressFamily.InterNetwork,
       SocketType.Dgram, ProtocolType.Udp);
    ```

LINKÖPINGS UNIVERSITET

AddressFamily Enum https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.addressfamily?view=netframework-4.7.2

Specifies the addressing scheme that an instance of the Socket class can use.

InterNetwork **Address for IP version 4.**

InterNetworkV6 **Address for IP version 6.**

SocketType Enum

https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.sockettype?view=netframework-4.7.2

- Specifies the type of socket that an instance of the Socket class represents.
- If you try to create a Socket with an incompatible combination, Socket throws a SocketException.

LINKÖPINGS UNIVERSITET

ProtocolType Enum

https://docs.microsoft.com/en-us/dotnet/api/system.net.sockets.protocoltype?view=netframework-4.7.2

- Specifies the type of socket that an instance of the Socket class represents.
- If you try to create a Socket with an incompatible combination, Socket throws a SocketException.

Using Client Sockets https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-client-sockets

- A data pipe between your application and the remote device must be created before initiating a conversation.
- TCP/IP uses a network address and a service port number to uniquely identify a service which is combined called EndPoint .
  - `IPEndPoint ipe = new IPEndPoint(ipAddress,11000);`
- Dns.Resolve method queries a DNS server to map a user-friendly domain name (such as "host.contoso.com") to a numeric Internet address (such as 192.168.1.1).
  - ```
    IPHostEntry ipHostInfo = Dns.Resolve("host.contoso.com");
    IPAddress ipAddress = ipHostInfo.AddressList[0];
    ```

LINKÖPINGS UNIVERSITET
    ```

Synchronous Communication

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-client-sockets

```
try {
    s.Connect(ipe);
} catch(ArgumentNullException ae) {
    Console.WriteLine("ArgumentNullException : {0}", ae.ToString());
} catch(SocketException se) {
    Console.WriteLine("SocketException : {0}", se.ToString());
} catch(Exception e) {
    Console.WriteLine("Unexpected exception : {0}", e.ToString());
}
```

Synchronous Communication

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-a-synchronous-client-socket

- Suspends execution.
- Not suitable for heavy usage **if not used in a separate thread.**
- Use Send and SendTo which receive a byte stream.
  - ```
    byte[] msg = System.Text.Encoding.ASCII.GetBytes("This is a test");
    int bytesSent = s.Send(msg);
    ```
- Call `Shutdown()` and then `close()` method in the end to release both sockets.
  - ```
    s.Shutdown(SocketShutdown.Both);
    s.Close();
    ```

LINKÖPINGS UNIVERSITET

Asynchronous Communication

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-an-asynchronous-client-socket

- Does not suspend execution.
- Creates a thread in background.
- Suitable for heavy usage.
- Requires a callback which is called when the response from the server is available.
- Use BeginConnect()
  - ```
    public static void Connect(EndPoint remoteEP, Socket client) {
        client.BeginConnect(remoteEP,
            new AsyncCallback(ConnectCallback), client );

        connectDone.WaitOne();  //ManualResetEvent
    }
    ```
-
    ```

Asynchronous Communication

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-an-asynchronous-client-socket

```
private static void ConnectCallback(IAsyncResult ar) {
    try {
        // Retrieve the socket from the state object.
        Socket client = (Socket) ar.AsyncState;

        // Complete the connection.
        client.EndConnect(ar);

        Console.WriteLine("Socket connected to {0}",
            client.RemoteEndPoint.ToString());

        // Signal that the connection has been made.
        connectDone.Set();
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}
```

LINKÖPINGS UNIVERSITET

Asynchronous Communication

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-an-asynchronous-client-socket

```
private static void Send(Socket client, String data) {
    // Convert the string data to byte data using ASCII encoding.
    byte[] byteData = Encoding.ASCII.GetBytes(data);

    // Begin sending the data to the remote device.
    client.BeginSend(byteData, 0, byteData.Length, SocketFlags.None,
        new AsyncCallback(SendCallback), client);
}
```

Asynchronous Communication

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-an-asynchronous-client-socket

```
private static void SendCallback(IAsyncResult ar) {
    try {
        // Retrieve the socket from the state object.
        Socket client = (Socket) ar.AsyncState;

        // Complete sending the data to the remote device.
        int bytesSent = client.EndSend(ar);
        Console.WriteLine("Sent {0} bytes to server.", bytesSent);

        // Signal that all bytes have been sent.
        sendDone.Set();
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}
```

LINKÖPINGS UNIVERSITET

Asynchronous Communication

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-an-asynchronous-client-socket

```
private static void SendCallback(IAsyncResult ar) {
    try {
        // Retrieve the socket from the state object.
        Socket client = (Socket) ar.AsyncState;

        // Complete sending the data to the remote device.
        int bytesSent = client.EndSend(ar);
        Console.WriteLine("Sent {0} bytes to server.", bytesSent);

        // Signal that all bytes have been sent.
        sendDone.Set();
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}
```

LINKÖPINGS UNIVERSITET

Asynchronous Communication

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-an-asynchronous-client-socket

```
public class StateObject {
    // Client socket.
    public Socket workSocket = null;
    // Size of receive buffer.
    public const int BufferSize = 256;
    // Receive buffer.
    public byte[] buffer = new byte[BufferSize];
    // Received data string.
    public StringBuilder sb = new StringBuilder();
}
```

LINKÖPINGS UNIVERSITET

Asynchronous Communication

https://docs.microsoft.com/en-us/dotnet/framework/network-programming/using-an-asynchronous-client-socket

```
private static void Receive(Socket client) {
    try {
        // Create the state object.
        StateObject state = new StateObject();
        state.workSocket = client;

        // Begin receiving the data from the remote device.
        client.BeginReceive( state.buffer, 0, StateObject.BufferSize, 0,
            new AsyncCallback(ReceiveCallback), state);
    } catch (Exception e) {
        Console.WriteLine(e.ToString());
    }
}
```

LINKÖPINGS UNIVERSITET

Thanks for listening!

LINKÖPINGS UNIVERSITET