

Windows SDK

- Based on functions
 - Not object oriented
- Can make object oriented programs by creating objects that encapsulate functions
- Uses normally C and C++....

Windows messages

- The windows OS is based on messages
 - Sends messages to programs when events occur
 - Messages are out into the message queue
 - Message loop in program retrieves messages and handles the message
- Message handling example
 1. Mouse click
 2. Mouse driver
 3. I/O manager
 4. User32
 5. Message queue
 6. WinMain
 7. User32
 8. WndProc

Pointers, SDK Deci(?)

Compact C/C++ notations

```
int tab[] = {0,1,2,3,5,8,-1}
int* pi = tab;
while({}*++pi != -1){
    cout << *pi << ',';
}
cout << endl;
```

Pointers and functions

```
#include <iostream>
using namespace std;

void Double(int* pi)
{
    int tall = *pi;
    tall = tall * 2;
    *pi = tall;
}

int main()
```

```
{
    int n;
    cout << "Enter n : ";
    cin >> n;
    Double(&n);
    cout << "n x 2 = : " << n << endl;
}
```

Dynamic variables must always be deleted

```
int main(){
    X a("Hello");
    X* pb= new X("World");

    a.Print();
    pb -> Print();
    delete pb;
}
```

C#

```
class HelloWorld{
    public static void main(){
        System.Console.WriteLine("Hello World");
    }
}
```

Compile

```
csc /target:exe helloworld.cs
```

.NET components

- Common language runtime(CLR)
 - Kernel, Required to run CIL code
- Framework class library(FCL)
 - Contains 6000+ classes for use in .NET library(replaces API calls)
- Compilers
 - Create assemblies containing CIL to run on the CLR

Namespaces

- A namespace is a way to organize .NET classes
- A namespace groups together classes with similar functions

- A namespace also makes it easier to avoid duplicate class names
- All the classes in the FCL are divided into namespaces based on their functionality
- A class is identified by both its namespace and actual name
- When we use a class, we need to tell the compiler both the namespace and the name of the class
- Examples:
 - `System.Console.WriteLine("Hello World");`
 - `Library.Book book = new Library.Book("Title");`

Namespaces - Importing

- For commonly used namespaces, we can import the namespace by using the using directive
- Example:

```
using System;
class CelloWorld {
    public static void Main(){
        Console.WriteLine("Cello World");
    }
}
```

Namespaces - creating

- A namespace is created by wrapping our class(es) in a namespace block

Namespaces - nesting

- Namespaces can be nested
- The using keyword does not provide access to namespaces nested below the specified level

Namespaces - aliases

- One can use aliases to provide a simplified name for a class or namespace

```
using s = System; // Namespace alias
using c = System.Console; // Class alias

s::Console.WriteLine();
c.WriteLine();
```

References

- References points to external libraries
- By default, not everything is referenced from a project.
- Might need to set up a reference to access a given namespace.
- References can be browsed in the object browser in VS

Managed Code

- Managed code is code that targets the CLR
- Code that does not target the CLR(old DLL's...) is unmanaged code
- Managed code is handled by the CLR, that provides
 - Security checks
 - Memory Management/Garbage Collection
 - Access to managed data

Assemblies

- An assembly is a .NET library or executable with the following properties:
 - Contains IL code which is compiled .NET source for example C# or VB.NET
 - Contains metadata which describes the interface.

Libraries

Value Types vs Reference types

- .NET has two kind of types
 - Value Types
 - Reference types

Reference types

- Reference types are stored on the heap
- A pointer to the memory location occupied by the type is store on the stack

Value types

- A value type does not point to the heap, it kepps its value on the stack*
- When assigned to a new variable, the content is copied
- Still behaves as an Object, and has methods
- Implicit inheretance from System.ValueType

Boxing

- Boxing is the process that happens then a Value Type is encapsulated in a Reference Type, and placed on the stack
- This creates a new copy of the value type
- Using a non-generics data structudre causes all Value Types to be boxed(And needs to be unboxed upon retrieval)
- Boxing is a resource intensive
- When using a data structure that supports generics, Boxing does not take place

REP: Common intermediate Language

- Visual Studio does not let us mix different languages in the same VS project
- But it is possible to create different projects inside a solution, where each project uses its own language
- It is also possible to manually assemble the .NET modules into .NET assemblies

- All .NET code no matter the language are compile to CIL code
- Possible to use several different .NET languages in the same project
- One language can use or subclass classes written in another language
- One language can throw an exception that is caught in another language

Subclasses

- As seem subclasses across languages
- This is written the exact same way as you would extend a native class
- Just remember to import the required namespaces(There might be implicit namespaces in the foreign language)

Compiling

- Normally, when code is compiled with a .NET compiler, we get a standalone assembly, which contains
 - Metadata
 - Executable CIL code
- Linking, as known from C++ is normally no longer done manually, but done automatically at runtime.

Demo - IL code similarity

- IL code output will be similar for similar code, no matter the original language
- Differences are due to compiler handling
- Note that some languages might lack some features, so writing identical code in two languages might not always be possible

What is software architecture

- Create a design that ensures
 - Usability
 - Business requirements
 - Performance
 - Stability
 - Maintainability
 - Bug fixing
 - Changing requirements
 - Security
- Three participants
 - User
 - Business
 - System
- Conflicts
 - Tradeoffs
- Must identify and consider key scenarios and quality attributes

Key Architecture Principles

- Build to change instead of build to last
- Model to analyze and reduce risk

- Use models and visualizations as a communication and collaboration tool
- Identify key engineering decisions

Delegate

- A delegate is a list of function pointers
- When called, a delegate will call all the function pointers it contains
- A delegate can be used to control program flow dynamically, since pointers can be added or removed at runtime
- Allows for a clear separation of modules, the owner of the delegate does not need to know about or depend upon the modules subscribing to the delegate

Patterns

- A pattern is a predefined solution to a common problem
- Patterns are found in all areas of creativity - Architecture, electronics, crafts, software design, software architecture
- Documented and published

Architectural Patterns

- Also referred to as Architectural Styles
- Provides an abstract framework
- ...

Client/Server

- Distributed system
 - Both the client and the server holds part of the application
- Clients are typically concerned about UI, while the server takes care of the business logic
- Clients make requests, server fulfills them
- Multiple clients may use one server

Scenarios

- Web application
- Mail/messaging systems
- Collaboration software
- Banking system
- Remote database access
 - Always?

Advantages

- Security
- Centralization data access
- Ease of maintenance

Disadvantages

- Scalability
- Stability

What is LinQ?

- Language Integrated Query
- LinQ is a Query language designed to query in memory data, databases, XML and other data sources
- Similar to SQL
- Works with all data structures implementing the IEnumerable interface

```
public class test
{

    static void Main(string[] args)
    {

        List<String> ls = new List<String>()
        {
            "Bob",
            "Alice",
            "Oscar",
            "Mike"
        };
        foreach(var x in ls.OrderBy(x=> x))
        {
            Console.WriteLine(x);
        }

        Console.ReadKey();
    }
}
```

IEnumerable

- A collection class implementing this interface can be iterated over with a foreach statement, as well as queried by something
- GetEnumerator uses the **yield** keyword to return elements one by one
- Normally done inside a loop
- Looks like a return statement, but does not end the method on invocation

```
namespace IEnumerable
{
    public class SimpleCollector<T> : IEnumerable<T> // Vil automatisk lage noen
    metoder, kopierer de fra forelesning som har noe å si
    {
        private list<T> data = new List<T>();

        public void Add(T item)
```

```
{
    data.Add(item);
}

public T Get(T index)
{
    return data.ElementAt(index);
}

public IEnumerator<T> GetEnumerator()
{
    foreach(T x in data)
    {
        yield return x;
    }
}
}

class Program
{
    static void Main(string[] args)
    {
        SimpleCollection<string> sc = new SimpleCollection<string>();

        sc.Add("Hei");
        sc.Add("Hello");

        foreach(string s in sc)
        {
            Console.WriteLine(s); // Will cause errors due to SimpleCollection list
being private
        }

        Console.ReadKey();

    }
}
}
```

Using LINQ

- To use LINQ we must include the System.Linq namespace
- This adds appropriate extension methods to all classes implementing the IEnumerable interface
- These extension methods provides functionality for running queries on the collection object
- Some important methods:
 - Select
 - Where
 - OrderBy
 - GroupBy
 - Join

The `DbSet<T>` class

- The `DbSet<T>` class is a collection of entity classes, and represents a physical table in your database
- A table can be loaded from the database simply by calling the `Set<>` method on the `DbContext`

Step 1 - We need entity classes

- Entity classes can be manually created as outlined previously using ADO.NET
- Or they can be automatically generated by Visual Studio

ASP.NET

- ASP.NET uses .NET technology to create web application
- Can theoretically be written in any .NET language, but usually c# or VB.NET is used
- Uses HTML for presentation

ASP.NET API's

- Web Forms
- MVC
- Web API

ASP Tags

- The HTML file uses special `<asp>` tags to represent .NET controls and other special elements.
- In code, these behaves just like ordinary .NET controls
- These are preprocessed server-side before the HTML code is sent to the client

Testing and Debugging

- Testing and debugging is done using the built in webserver in Visual Studio
- You can use breakpoints and watches just as with a local application
- Note that the browser might time out during a debugging session. This doesn't affect the debugging session itself, but you would not get any new data

Microsoft MVC

- Based on ASP.NET
- Uses the familiar Model-View-Controller pattern
 - Separates data from logic from presentation
- Uses Razor for creating dynamic html pages
 - Code blocks and inline expressions are indicated by @
 - Code blocks can be C# or VB
- Uses routing to determine which controller and method will handle a request
 - Decoupled, easy to add additional functionality

WebAPI

- ADO.NET API for making restful webservices
 - REST(Representational State Transfer) is an architectural style

- Based on stateless operation
- Decoupled architecture
- Not limited to a fixed data format, but JSON is often used
- URI's contain resource identification
- BASED on HTTP, uses HTTP verbs
- Built on top of the Microsoft MVC web platform
- Serves data in the format requested by the client(XML or JSON, for HTML use MVC)
 - Marshalled automatically

Universal Windows Platform(UWP)

- Platform built on top of the Windows Runtime(WinRT)
- Designed to run on all modern Windows devices:
 - Windows 10
 - Xbox
 - Surface Hub
 - Hololens
 - Windows Phone/Mobile
- Can be developed in C#/VB/C++/JavaScript
- Does not have to be .NET
- Does not support the full .NET ecosystem
- Not intended for non-Windows OS
 - Use Xamarin
- Distributed via Microsoft Store
 - Also sideload support
- UI options similar to WPF
- Secure Execution
- Limited support for ADO.NET
- Optimized for touch scenarios

Lambda Expressions

- While normal expressions return a value, a lambda expression will return a method
- This is particularly useful when designing anonymous methods to use with delegates/events
- Lambda expressions allow us to bring functional programming into .NET

Defining Lambda Expression

- A lambda expression has these elements
 - A list of parameters
 - The `=>` operator
 - The method body (Or a simple expression)
- `(param) => {body;}`

Expression/Statement Lambda

- Lambda statements are divided into expression and statement lambdas

- An expression lambda is a one line statement, without brackets. The result of the expression is used as the return value.
- A Statement lambda contains a body in brackets, needed if the lambda is multiline. An explicit return statement is used to return a value.
- Note that the term lambda expression still refers to both types

Using lambda expressions

- Lambda expressions are pointers to functions
 - Not immediately evaluated
- Normally stored in a delegate
 - Custom defined delegate
 - `Func<in T1, ..., in Tn, out T>`
 - `Action<in T1, ..., in Tn>`

Sample lambda expressions

- `x => x*x`
- `x => {return x*x;}`
- `() => timer.start()`
- `(x,y) => {x += y; return x/y;}`
- `(int x) = x/2`

Out of scope variables

- A lambda expression can refer to a variable that is no longer in scope when the function is called
 - The expression will still have access to this variable
 - The variable will only be garbage collected when the delegate holding the lambda collected

Dynamic program flow

- Lambda expressions are great for controlling program flow
- They can be created in one part of the applications and passed to another
- This allows for more dynamic control of a program flow at runtime

Parallel LINQ (PLINQ)

- Many LINQ operations benefits from being run in parallel
- This allows the computer to put multiple threads at work processing the data
- Just start the LINQ chain with a call to `AsParallel()`
- Not all data/operations can be optimized for parallel processing
- Beware thread safety
- Many configuration options (parallelism, ordering, custom partitioning, etc)

Parallel Execution

- Parallel for loops:
 - Allows us to do multiple loop iterations in parallel instead of one by one sequentially
 - `Parallel.For(int, int, Action<int>)`

- `Parallel.ForEach(IEnumerable<t>, Action<t>)`
- Parallel delegates:
 - `Parallel.Invoke(Action[])`
 - Note: Operates on an array of delegatesm does not parallelize each individual delegate

Asynchronous Processing

- Running a method asynchronous

UI Design/UX/Universal Design

- Note that this is a big topic
 - We will only be looking at the surface
- **GOAL** - Usability
- The design should be
 - Easy to use by everyone
 - Easy to understand by everyone
 - Text should be understandable
 - Elements should be identifiable
 - Icons/images should be meaningful

Do your research

- When designing a user interface, ALWAYS research your target audience and map their needs and requirements based on principles for good UI design
- Do NOT give your own opinions much weight. An interface suited for a developer is not going to be well suited for the majority of users unless you design an IDE.
- DO test the UI on users
 - Mock ups is often a good start
 - Focus groups
- Consider getting professional help, UX is a whole field of study, not the job of a lone developer. UD is not simple either.

Meet Expectations

- When someone use a given computer system, they have expectations
 - Windows layout and design
 - Control placement
 - Key Bindings
 - Colors
- Comply with the standard look & feel on their platform
- Make sure to use system colors so that the color scheme of your application follows the user settings
- Make sure the application obeys changes to the system properties, like font size
- If the platform supports skinning, make sure your application also uses the skins properly

Visuals

- Remember that not everyone have 20/20 vision
 - Near/far-sighted, colorblind, blind, injuries, illness.

- By following the advice about meeting, something something
- Use Colors with high contrast values
- Remember color blindness
 - Not everyone sees all colors the same way
- Try to use standard window layout, for most programs, this means menu/toolbake at the top, dialog buttons at the bottom, og der forsvant tekste

Content

- Think about internationalization and Localization
 - This may even impact the design (ex: right-to-left readers)
 - Remember that texts may be longer in other languages than yours
- Beware the use of symbols
 - Not friendly to screen readers
 - Culturally dependent
 - Some symbols are taught, others are natural
 - What's natural to you might not be to others

Readability

- Don't try to put too much detail into one page/window so on

Sensible navigation

- People have different control preferences, like mouse, keyboard, touch, etc. Others have special needs
 - Let people use their preferred equipment.
- Make sure the navigation path through your application is sensible
 - TEST
- Make navigational elements obvious

Sound, Video and visuals

Robustness

- The design should be able to handle user errors gracefully, without breaking
- User errors should not have major consequences
 - Confirmatio
 - Undo
 - Autosave

Extension methods

- Extension methods allow us to add extra methods to an existing class without having access to the source code of the class
- This is heavily used by LINQ, adding methods to all classes implementing IEnumerable
- Only provides access to public properties/methods
- Really just a smart wrapper method, but is used like a real member, preventing all the usual problems with wrappers
- Implemented as a static method in a static class

- Method signature tells which Class/Interface it applies to(**this** keyword, first parameter)
- **Can Not**
 - Override existing methods
 - Access private/protected fields

LINQ to XML

- LINQ can handle XML files as a data source
- XML files can be bound to GUI elements
- But
 - XML is hierarchical - tree structure
 - XML is something else as well

XElement

- Core LINQ to XML class
- Represent a single XML element

LINQ to XML methods

- In LINQ to XML we work on lists(IEnumerable) of XElement (and Xattribute) objects
- **Load()** Loads an XML document from file
- **Parse()** Creates XML document from string
- **Save()** Saves the XML document to file
- **Descendants()/DescendantsAndFiles()**: returns the child element of an XElement
- **Ancestors()/AncestorsAndSelf()**:
- **Attribute()**: returns the attribute of an XElement

Join

- Joins can be performed across source types
 - EX: XML to EF
- Inner Join
 - The join operator in LINQ perform an inner join
 - Only returns rows where both match
- Left/Right Outer join
 - Returns all elements from one table and only matching elements from the other
 - No LINQ operator handles outer joins by default
 - Can be Accomplished by some clever LINQ statements
 - Or by implementing your own extension method
- Full outer joins
 - Returns all rows from tables and so on
- By default, LINQ checks to see if the two fields are equal
- More complicated checking can be done by writing your own comparer class using IEnumerable