C# - Summary

Class-Design

Fields

Fields store values within the class.
Allowed access-modifiers:

```
private ...     // Only accessable from within the class
protected ...   // Only accessable from own class and classes that inherit this class

// Following are also possible, but not recommended
public ...      // Accessable from everywhere
internal ...    // Only accessable from within the namespace
```

Basic field definition:

```
// Definition without assignments
private int myIntField;
private MyObject obj;
private string[] myStringArray;

// Definition with assignments
private float myFloatField = 0.45;
private double[] myDoubleArray = new double[5];
private long[] myLongArray = new long[] { 1, 4, 7 };
```

Advanced field definition:

```
// Static fields
private static String companyName;
```

Properties

Properties can get and set fields with additional checks, conditions or operations.

```
private int squareWidth;
private int xPosition;
private int yPosition;

// Property with set-condition
public int SquareWidth
{
    get { return squareWidth; }
    set
    {
        if(value >= 0) squareWidth= value;
        else squareWidth= 0;
    }
}

// Readonly-Property
public int Area
{
    get { return squareWidth * squareWidth; }
}
```

```
// Default Get/Set Property
public int XPosition
{
    get { return xPosition; }
    set { xPosition = value; }
}

// Alternative Style
public int YPosition
{
    get => yPosition;
    set => yPosition = value;
}

// Auto-Implemented Properties (no field required)
public int ZPosition { get; set; }
```

Methods

```
// Method without return value or Parameters
public void MyMethod()
{
    // ...
}

// Method with return value and one parameter
public int MyReturnMethod(int inParameter)
{
    // ...
    return outparameter;
}

// Method with optional parameters
public void MyOptionalParamMethod(int required, float optional = 1.5)
{
    // ...
}

// Static Method
public static void MyStaticMethod(float maxValue)
{
    // ...
}

// Overloading Methods
public double MyOverloadedMethod(int inputValue)
{
    // ...
    return outparameter;
}
public double MyOverloadedMethod(float inputValue)
{
    // ...
    return outparameter;
}
```

Indexer

An indexer is mostly like a property for an array:

Standard use for an indexer:

```
private int[] myArray= new int[] { 1, 4, 7, 12 };

public int this[int idx]
{
    set { myArray[idx] = value; }
    get { return myArray[idx]; }
}
```

Use indexer with keywords:

```
string[] days = { "Sun", "Mon", "Tues", "Wed", "Thurs", "Fri", "Sat" };

private int GetDay(string testDay)
{
    for (int j = 0; j < days.Length; j++)
    {
        if (days[j] == testDay) { return j; }
    }
}

public int this[string day]
{
    get { return (GetDay(day)); }
}
```

Constructor

```
class MyClass
{
    private int myField1;
    private int myField2;

    // Default Constructor
    public MyClass()
    {
    }

    // Overloading constructors
    public MyClass(int _myField1, int _myField2)
    {
        myField1 = _myField1;
        myField2 = _myField2;
    }

    // Extending constructors
    public MyClass(int _myField1):this(_myField1,0) { }
}
```

Overriding Methods

Methods can be overridden and the behaviour of the Method can be changed.

```
public override string ToString()
{
    // ...
}
```

Operation Overloading

Operators for classes can be overridden and givven new functionality.

```
public static int operator+ (int a, int b)
{
    // ...
    return c;
}
```

Enumerations

```
public enum MyEnum
{
    VAL1,
    VAL2,
    VAL3
};
```

Arrays & Lists

Definition of Arrays

```
// Creating array without assigning
private int[] myArray;

// Creating array with fixes lenght
private int[] myArray;
myArray = new int[3];
// OR
private int[] myArray = new int[3];

// Creating pre-filled array
private string[] myArray = new string[] { "Str1" , "Str2" , "Str3" };

// Alternative syntax
private string[] myArray = { "Str1" , "Str2" , "Str3" };

// Creating a 2-Dimentional array
private int[,] myMultiDimensionalArray = new int[2,3];

// Creating a jagged array
private int[][] myJaggedArray = new int[6][];
myJaggedArray[0] = new int[4];

// Filling an array made from objects
private MyObject[] objArray = new MyObject[5];

for(int i = 0 ; i < objArray.lenght ; i++)
{
    objArray[i] = new MyObject();
}
```

Definition of Lists

```
// Defining List
private List<int> myList = new List<int>();
```

```
myList.add(50);
```

Generic Datatypes

Creating Generic class

```
public class MyGenericClass<T>
{
    private T[] elements;
    private int count;

    public MyGenericClass(int size)
    {
        elements = new T[size];
    }
}
```

Interfaces

Creating an Interface

```
public interface IMyInterface
{
    void MethodThatMustBeImplemented(string param);
}
```

Assigning an Interface

```
// Assigning to a default class
public class MyClass : IMyInterface, IMyOtherInterface
{
    // ...
}

// Assigning to a generic class
public class MyClass<T> where T : IComparable
{
    // ...
}
```

IComparable Interface

```
class MyClass : IComparable
{
    public int CompareTo(object obj)
    {
        if( ... ) return -1; // objX is bigger than this object
        if( ... ) return 1; // objX is smaller than this object
        return 0;   // both objects are the same
```

```
    }
}
```

IDisposable Interface

```
class MyClass : IDisposable
{
    private bool isDisposed = false;

    ~MyClass
    {
        // Deconstructor actions
        Dispose();
    }

    public void Dispose()
    {
        if(!isDisposed)
        {
            isDisposed = true;
            // e.g. objectCounter--;
            GC.SuppressFinalize(this);
        }
    }
}
```

Exception Handling

Basic Exception Functions

```
// Throwing Exception
throw new Exception("Error-Message");

// Force exception-test
checked { a *= b }

// try-catch-finally
try
{
    // Try Executing code
}
catch (IndexOutOfRangeException ioorEx)
{
    // Catch an Index-Out-Of-Range exception
}
catch(Exception ex)
{
    // Catch other exceptions
}
finally
{
    // Execute functions that must be done after try as well as catch.
    // E.g. Saving file, Closing SQL-Connection, ...
}
```

Exception Types

The following list shows some exception-types of the .NET Framework.

```
Exception
SystemException
IndexOutOfRangeException
NullReferenceException
AccessViolationException
InvalidOperationException
ArgumentException
ArgumentNullException
ArgumentOutOfRangeException
ExternalException
COMException
SEHException
```

Inheritance

Access-Keywords

By using the `protected` -Keyword, the fields can be accessed by the cild-classes.

```
protected int xPosition;
```

Inheriting a class

Inherit everything from the parent-class:

```
class MyChildClass : MyParentClass
{
    // ...
}
```

Abstract

Abstract Class: Can not be created as an object. Used as parent-class
for child-classes. E.g.: Shape-Class -> child: Rectangle.

```
abstract class MyBaseClass
{
    // ...
}
```

Abstract Method: Must be implemented in every child-class.

```
public abstract int MyAbstractMethod();
```

Virtual

Declared in Parent-Class. Allows for the method to be overridden.

```
public virtual int MyVirtualMethod()
{
    // ...
}
```

Sealed

The `sealed` keyword prevents the method from beeing overridden.

```
public sealed int MySealedMethod()
{
    // ...
}
```

Override

Overrides the methods declaration in the parent-class.

```
public override int MyVirtualMethod()
{
    // ...
}
```

To keep all functions from the parent-method and add more functions, use the `base` keyword:

```
public override int MyVirtualMethod()
{
    base.MyVirtualMethod();
    // ...
}
```

Constructors

To Inherit the functionality of the parent-constructor, use the `base` -Keyword:

```
public MyChildClass(int _x, int _y, int _r): base(_x, _y)
{
    // ...
}
```

Unit-Tests

Unit-Test Basics

Naming Test-Methods:

- The Method that should be tested
- The scenario that should be tested
- The expected result

Parts of a Test-Method

- Arrange
  – Create the start-conditions
- Act
  – Execute the test
- Assert
  – Check the result

Defining a Test-Class & Test Methods

```
[TestClass]
public class UnitTest
{
    // ...
}

[TestMethod]
public void MyTestMethod()
{
    // ...
}
```

Calling Test-Method with DataRows

```
[DataTestMethod]
[DataRow( "funcParam1" , "funcParam2" )]
[DataRow( "FUNCPARAM1" , "FUNCPARAM2" )]
[DataRow( "funcparam1" , "funcparam2" )]
public void MyTestMethod(string param1, string param2)
{
    // ...
}
```

Calling Test-Method with expected exception

```
[TestMethod]
[ExpectedException(typeof(ApplicationException))]
public void MyTestMethod()
{
    // ...
}
```

Asserting Results

```
Assert.IsTrue(myBooleanValue);
Assert.AreEqual(myObject1, myObject2);
Assert.IsNotNull(myObject);
```

Checking float-values

```
const double delta = 1e-04; // 4 floating points
```

```
Assert.AreEqual(myFloatValue1, myFloatValue2, delta);
```

SQL - ODBC

Connection-Strings

```
// Connection-String MS-Access
string conStr = @"Driver={Microsoft Access Driver (*.mdb, *.accdb)};Dbq=" + Application.StartupPath + @"\MyDataba
```

```
// Connection-String MySQL
string conStr = @"Driver={MySQL ODBC 3.51 Driver};Server=IP-or-Adresse;Database=dbname;User=dbuser;Password=dbpas
```

Basic SQL-Class

The following class is a simple ODBC-Class for basic SQL-Functions

```
class SQLC
{
    private static OdbcConnection connection = null;

    // Open and close connections within the program with
    // SQLC.Connection.Open(); or SQLC.Connection.Close();
    public static OdbcConnection Connection
    {
        get
        {
            if(connection == null)
            {
                string conStr = @"Driver={...}";
                connection = new OdbcConnection(conStr);
            }
            return connection;
        }
    }

    // Get the result-set of an SQL-Query
    public static OdbcDataReader ExecuteQuery(string sql, params object[] vals)
    {
        OdbcCommand cmd = new OdbcCommand(sql,Connection);
        foreach(object parameter in vals) cmd.Parameters.AddWithValue("",parameter);
        OdbcDataReader reader = cmd.ExecuteReader();
        return reader;
    }

    // Execute a Non-Query
    public static int ExecuteNonQuery(string sql, params object[] vals)
    {
        OdbcCommand cmd = new OdbcCommand(sql,Connection);
        foreach(object parameter in vals) cmd.Parameters.AddWithValue("",parameter);
        Connection.Open();
        int result = cmd.ExecuteNonQuery();
        Connection.Close();
        return result;
    }

    // Execute a Scalar
    public static object ExecuteScalar(string sql, params object[] vals)
    {
        object o;
        OdbcCommand cmd = new OdbcCommand(sql,Connection);
        foreach(object parameter in vals) cmd.Parameters.AddWithValue("",parameter);
```

```
        Connection.Open();
        o = cmd.ExecuteScalar();
        Connection.Close();
        return o;
    }
}
```

Usage of SQLC-Class

Using the `ExecuteQuery()` Method:

```
SQLC.Connection.Open();
OdbcDataReader reader = SQLC.ExecuteQuery("SELECT ...");
while(reader.Read()) lbxList.Items.Add(reader["tableColumn"]);
SQLC.Connection.Close();
```

Using the `ExecuteNonQuery()` Method:

```
SQLC.ExecuteNonQuery("UPDATE ...");
```

Using the `ExecuteScalar()` Method:

```
object o = SQLC.ExecuteScalar("SELECT ...");
if(o != null) txb.Text = o.ToString();
```

Data-Handling with SQLC-Class

Creating a DataTable:

```
public static DataTable CreateDT(string sql)
{
    OdbcDataAdapter da = new OdbcDataAdapter(sql,Connection);
    DataTable dt = new DataTable();
    da.Fill(dt);
    return dt;
}
```

Using the DataTable-Object for Listboxes:

```
DataTable dt = SQLC.CreateDT("SELECT ...");
lbx.DisplayMemeber = "Column1";
lbx.ValueMember = "Column2";
lbx.DataSource = dt;
```

Using the DataTable-Object for DataGridViews:

```
DataTable dt = SQLC.CreateDT("SELECT ...");
dgv.DataSource = dt;
```

SQL-Transactions

```
OdbcTransaction trans = null;
try
{
    SQLC.Connection.Open();
```

```
    trans = SQLC.Connection.BeginTransaction();
    cmd.Transaction = trans;
    // Execute SQL-Statements here
    trans.Commit();
}
catch(Exception ex) trans.Rollback();
finally SQLC.Connection.Close();
```

Classes

String

```
string myStr = "This is a test String";

// Analyse and manipulate Strings
myStr.StartsWith("This");   // -> true
myStr.Split(" ");           // -> ["This","is","a","test","String"]
myStr.SubString(6);         // -> "s a test String"
myStr.SubString(6,7);       // -> "s a tes"
myStr.Replace(' ','-');     // -> "This-is-a-test-String"

// Haystack-Needle
if(hayStack.IndexOf(needle) != -1)
{
    // Found!
}
else
{
    // Not Found!
}
```

StringBuilder

```
StringBuilder sb = new StringBuilder();

sb.Append("Text 1 ");
sb.AppendFormat("Text {0}", 1);

string finalString = sb.ToString();
```

StreamReader

```
StreamReader sr = new StreamReader("filePath.txt");
while(sr.Peek() >= 0)
{
    outString = sr.readLine();
}
sr.Close();
```

```
StreamReader sr = new StreamReader("filePath.txt");
while((outString = sr.ReadLine()) != null)
{
    // ...
```

```
}
sr.Close();
```

StreamWriter

```
StreamWriter sw = new StreamWriter ("filePath.txt");

string myName = "Tom"
sw.writeLine("I am {0}", myName);

sw.Close();
```

Math

```
// Usefull Math-Class Methods
Math.PI;            // -> 3.141...
Math.Abs(-5);       // -> 5
Math.Sqrt(16);      // -> 4
Math.Pow(8,2);      // -> 64
Math.Sin(82);       // -> sin(82)
Math.Cos(39);       // -> cos(39)
Math.Round(4.8);    // -> 5
```

Random

```
Random rnd = new Random();
int randomInt = rnd.Next(0,100);
```

Forms & Dialogs

Message-Boxes

Show a message-box

```
MessageBox.Show("Message","Title",MessageBoxButton.OK);
```

Get output from message-box

```
if(MessageBox.Show("Content","Title",MessageBoxButtons.OKCancel) == DialogResult.OK)
{
    // ...
}
```

Read Key-Inputs

```
e.KeyChar == 13;
```

Close a Form

```
this.Close();
```

Open File Dialog

```
// Open File with Dialog
ofdOpen.InitialDirectory = Enviroment.GetFolderPath(Enviroment.SpecialFolder.Desktop);

if(ofdOpen.ShowDialog() == DialogResult.OK)
{
    StreamReader sr = new StreamReader(ofdOpen.Filename);
    while((outString = sr.ReadLine()) != null)
    {
        // ...
    }
    sr.Close();
    MessageBox("Loading Successfull");
}
```

Data Handling

Data Row View

```
// E.g.: Extract Data from Listbox
DataRowView drv = (DataRowView) lbx.SelectedItem;
string value1 = drv["Column1"].ToString();
string value2 = drv["Column2"].ToString();
```

DataGrid

```
// Create the DataTable-object
DataTable dtGrid = new DataTable();
// Add Columns to Grid
dtGrid.Columns.Add("ColumnName1",typeof(int));
dtGrid.Columns.Add("ColumnName2",typeof(string));

// Fetch data for grid (e.g. from other grid)
DataTable dtDataRaw = (DataTable)(dgv.DataSource)

// Add rows to grid
DataRow newRow;
newRow = dtGrid.NewRow();
newRow["Column1"] = 5;
newRow["Column2"] = "Content";

// Add row to grid
dtGrid.Rows.Add(newRow);

// Add data to grid
dgv.DataSource = dtGrid;
```

Form Graphics

Graphics-Setup

```
// Move center to middle of screen
e.Graphics.TranslateTransform(pbx.Width / 2, pbx.Height / 2);

// Set cartesian coordinate system
e.Graphics.ScaleTransform(-1,1);

// Enable Anti-Aliasing
e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
```

Drawing Arrows

```
Point p1 = new Point(0,0);
Point p2 = new Point(8,6);
Pen pen = new Pen(Color.red,5);
AdjustableArrowCap aac = new AdjustableArrowCap(5,5,true);

pen.CustomEndCap = aac;
pbx.Invalidate();
e.Graphics.DrawLines(pen, p1, p2);
```

Date handling / Date functions

Creating DateTime-Objects

```
// Create a default DateTime-Object with the value `01-01-0001 00:00:00`
DateTime dt = new DateTime();

// Create DateTime-Object with current time
DateTime dt = DateTime.Now;

// Create DateTime-Object with set date
DateTime dt = new DateTime(1999, 11, 16);

// Create DateTime-Object with set date and time
DateTime dt = new DateTime(1999, 11, 16, 18, 42, 0);
```

Output DateTime-Objects

```
// Pre-Defined methods
Console.WriteLine(DateTime.Now.ToShortDateString());
Console.WriteLine(DateTime.Now.ToLongDateString());
```

```
// Region-Specific output
var culture = new System.Globalization.CultureInfo("en-US");
Console.WriteLine(DateTime.Now.ToString(culture.DateTimeFormat));
```

```
// Custom formats
DateTime dt = new DateTime(2042, 12, 24, 18, 42, 0);

Console.WriteLine(dt.ToString("MM'/'dd yyyy"));     // > 12/24 2042
Console.WriteLine(dt.ToString("dd.MM.yyyy"));       // > 24.12.2042
Console.WriteLine(dt.ToString("MM.dd.yyyy HH:mm")); // > 12.24.2042 18:42
Console.WriteLine(dt.ToString("dddd, MMMM (yyyy): HH:mm:ss")); // > Wednesday, december (2042)
```

```
Console.WriteLine(dt.ToString("dddd @ hh:mm tt",
System.Globalization.CultureInfo.InvariantCulture)); // > Wednesday @ 06:42 PM
```

File handling

Combine path and filename

```
string fileName = "test.txt";
string path = @"C:\Users\Public\TestFolder";

string filePath = System.IO.Path.Combine(path, fileName);
```

Copy a file

```
// Copy 1 file
if (!System.IO.Directory.Exists(targetPath))
{
    System.IO.Directory.CreateDirectory(targetPath);
}

System.IO.File.Copy(sourceFile, destFile, true);
```

```
// Copy a directory
if (System.IO.Directory.Exists(sourcePath) && System.IO.Directory.Exists(targetPath))
{
    string[] files = System.IO.Directory.GetFiles(sourcePath);

    foreach (string s in files)
    {
        fileName = System.IO.Path.GetFileName(s);
        destFile = System.IO.Path.Combine(targetPath, fileName);
        System.IO.File.Copy(s, destFile, true);
    }
}
```

Rename / Move a file

```
// Move a file
string sourceFile = @"C:\Users\Public\public\test.txt";
string destinationFile = @"C:\Users\Public\private\test.txt";

System.IO.File.Move(sourceFile, destinationFile);
```

```
// Move a directory
string sourceDir = @"C:\Users\Public\public\test";
string destinationDir = @"C:\Users\Public\private";

System.IO.Directory.Move(sourceDir, destinationDir);
```

Delete a file

```
// Delete a file
if(System.IO.File.Exists(@"C:\Users\Public\DeleteTest\test.txt"))
{
    try
    {
        System.IO.File.Delete(@"C:\Users\Public\DeleteTest\test.txt");
    }
    catch (System.IO.IOException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

```
// Delete directory and all subdirectories
if(System.IO.Directory.Exists(@"C:\Users\Public\DeleteTest"))
{
    try
    {
        System.IO.Directory.Delete(@"C:\Users\Public\DeleteTest", true);
    }
    catch (System.IO.IOException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Usefull functions

C# Properties

```
// Setting Property
Properties.Settings.Default.MyProperty = "My Value";

// Get Property
string str = Properties.Settings.Default.MyProperty;
```

String formating types

```
"\r\n"
// OR
Enviroment.NewLine
```

Remove duplicates from Array

```
for(int i = 0; i < myArray.lenght; i++)
{
    for(int j = i + 1; j < myArray.lenght; j++)
    {
        if(myArray[i]==myArray[j])
        {
            myArray.RemoveAt(j);
            j--;
        }
    }
}
```

Culture Info

```
var culture = new System.Globalization.CultureInfo("de-AT");
```

Usefull Culture Coutry-Codes

Country	Code
German (Germany)	de-DE
German (Austria)	de-AT
German (Switzerland)	de-CH
English (UK)	en-GB
English (USA)	en-US
English (Australia)	en-AU

Debuging

```
Debug.print("Test-Output");
```