Applikationsudvikling II

Lecture 2

Andrea Valente
aval@sdu.dk
https://portal.findresearcher.sdu.dk/en/persons/aval

Topics:

- Chpt 5 – visual studio debugger
  - Task about debugging a C# program
- Chpt 6 – how to create classes

A simple program using a List

- ”ACTIVITY 5-2. USING THE DEBUGGING FEATURES OF VS”, page 77

Let’s:

- Create a project,
- Past the code in the activity (or in file code\activity_5_2.cs)
- Compile+run to **see what the code does…**

- **What do you think the code does/is supposed to do?**

Debug

```
1  class Program
2  {
3      static List<int> numList = new List<int>();
       0 references
       static void Main(string[] args)
```

Quick Actions and Refactorings...	Ctrl+.
Rename...	Ctrl+R, Ctrl+R
Remove and Sort Usings	Ctrl+R, Ctrl+G
Peek Definition	Alt+F12
Go To Definition	F12
Go To Base	Alt+Home
Go To Implementation	Ctrl+F12
Find All References	Shift+F12
View Call Hierarchy	Ctrl+K, Ctrl+T
Track Value Source	
Create Unit Tests	
Breakpoint	
Run To Cursor	Ctrl+F10
Force Run To Cursor	
Execute in Interactive	Ctrl+E, Ctrl+E
Snippet	
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Annotation	
Outlining	

Insert Breakpoint	
Insert Conditional Breakpoint	
Insert Tracepoint	
Insert Temporary Breakpoint	Shift+Alt+F9, T
Insert Dependent Breakpoint	Shift+Alt+F9, D

Show output from: Build

Insert breakpoint

```
namespace Activity5_2
{
    class Program
    {
        static List<int> numList = new List<int>();
        static void Main(string[] args)
        {
            LoadList(10);
            foreach(int i in numList)
            {
                Console.WriteLine(i);
            }
            Console.ReadLine();
        }
```

Use *step into* and then *step over* to inspect LoadList(10) function call!

Create & use a conditional breakpoint

```
10          }
11          Console.ReadLine();
12      }
        1 reference
13      static void LoadList(int iMax)
14      {
15          for (int i = 1; i <= iMax; i++)
16          {
17              numList.Add(i);
```

Location: Program.cs, Line: 17, Character: 13, Must match source

☑ Conditions

Conditional Expression | Is true | i==3

Add condition

Re-start

```
17              numList.Add(i);   ≤ 1ms elapsed
18          }
19      }
20  }
```

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
i	3	int
iMax	10	int
numList	Count = 2 View	System.Collections....

It stops automatically when the condition is true

By clicking on a variable you can inspect its value(s)

And questions about *List<>* ?

Break

- Questions so far?

How to create classes

*<< We will work with a simple **Employee** class.*

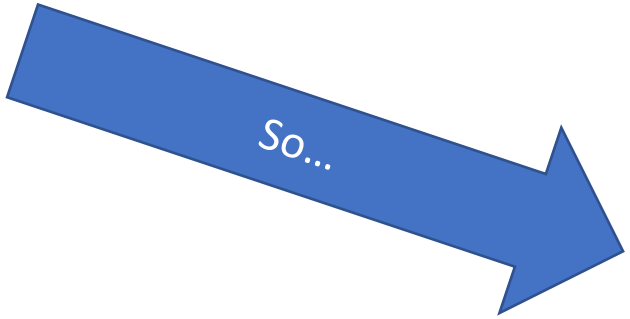
The Employee class will have properties and methods that encapsulate and work with employee data as part of a fictitious human resources application >>

- Questions to ask yourself:
  - what attributes do I want/need in the class?
  - what should an Employee object be able to **DO** ? (methods)

Let's design the class, then implement it **together**

Attributes and methods

- An employee object represents a person's data in the system
- It could have the following attributes:
  - employeeId which can just be a number (integer)
  - loginName which should be a string
  - password, a string too
  - department, also a string
  - name, a string
- An object in OOP has a life cycle:
  - It is created
  - Initialized with some data
  - Used, by calling methods on it
  - And eventually deleted from memory



Attributes and **methods**

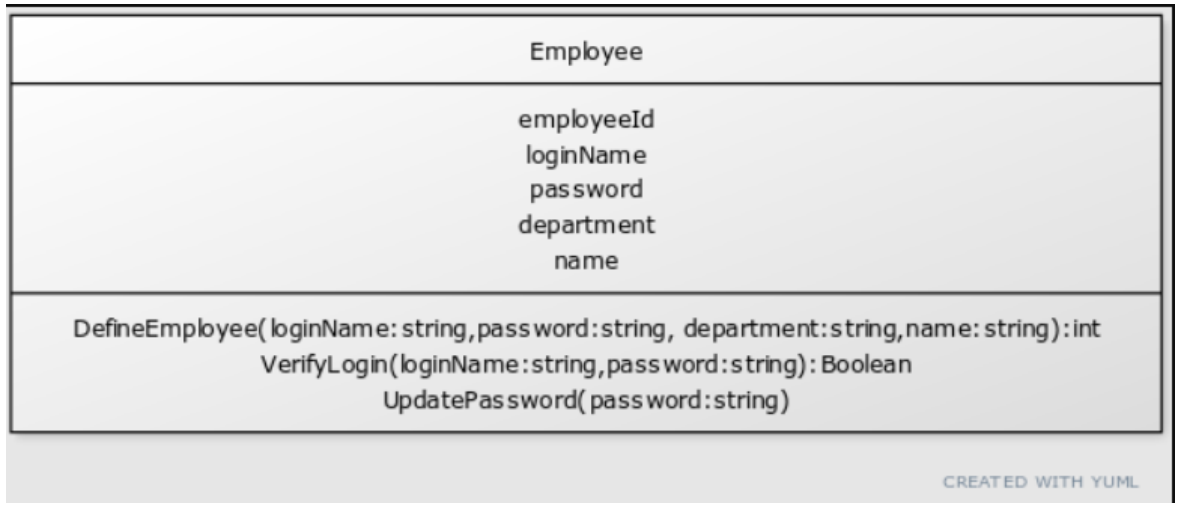
- I need methods to help me do some of these steps in the life cycle
- E.g.
  - **I can use *new* to create an employee object (done!)**
  - I need a method to initialize the data inside the object: *DefineEmployee*
  - I might want to check if the employee knows his/her login and pwd:
    - *VerifyLogin(login,pwd)* , and this method could reply true/false
  - A method to change the pwd for an employee: *UpdatePassword(newPwd)*
  - **How do I delete an object from memory in C#?**
    Do you know about this stuff? ;)

A UML class diagram would be useful here…

[Employee|employeeId;loginName;password;department;name|DefineEmployee(loginName:string,password:string

department:string,name:string):int;VerifyLogin(loginName:string,password:string):Boolean;UpdatePassword(password:string)]

https://yuml.me/diagram/scruffy/class/draw



Let’s create a new project …

- **… and implement the class Employee**
- We can proceed incrementally:
  - First just an ”empty” class,
  - Then we can add the attributes
  - And then the methods
  - Finally, we can write some code in the *main*, to create and use a few objects

- **Note:** we could also proceed **backwards!**
  - From the *main*, and use the intelligent IDE features to complete/auto-write the code
  - *Have you done this in some other projects?* **O_o**

Creating **Properties**

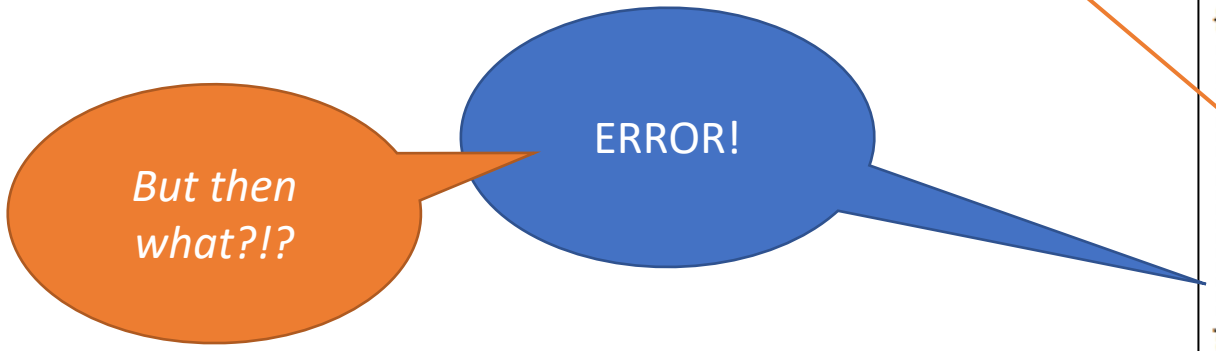
- Remember encapsulation? Here we want some of the attributes of an instance (AKA an object) to be impossible to access from the main (or other classes)…
  - <<The *private* **keyword** ensures that these instance variables can be manipulated only by the code inside the class>>

- Attributes names start with *lower case*,
  and a name starting with _ is usually a private attribute.

- Sure, but I do I access/initialize an attribute that is **private**??

```
class Employee
{
    private int _empID;
    private string _loginName;
    private string _password;
    private string _department;
    private string _name;
}

0 references
internal class Program
{
    0 references
    static void Main(string[] args)
    {
        Employee emp = new Employee();
        emp._empId = 1;
        Console.WriteLine( emp );
    }
}
```



Define a ”property block”

<<

- When a user of the class (client code) needs to query or set the value of these instance variables, public properties are exposed to them.
- **Inside the property block of code are a *Get block* and a *Set block*:**
  - The Get block returns the value of the private instance variable to the user of the class. This code provides a readable property.
  - The Set block provides a write-enabled property; it passes a value sent in by the client code to the corresponding private instance variable

>>

- For example we can define a **property Name**, that is
  - readable and writable from “outside” the class,
  - and that is in fact just a public name of the attribute *_name*

```
public string Name
{
    get { return _name; }
    set { _name = value; }
}
```

```
static void Main(string[] args)
{
    Employee emp = new Employee();
    emp.Name = "Andrea";
    Console.WriteLine( emp.Name );
}
```

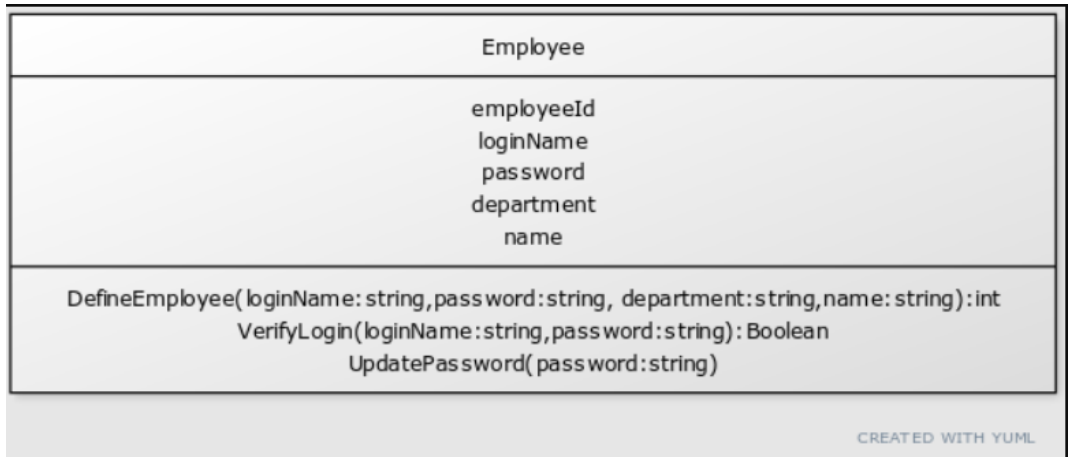
Validate

- We can even do **validation** using a propertie's Get and Set blocks

```
public string Password
{
    get { return _password; }
    set
    {
        if (value.Length >= 6)
        {
            _password = value;
        }
        else
        {
            throw new Exception("Password must be at least 6 characters");
        }
    }
}
```

Look at *code\Employee class_v3.cs*

Create methods



- Look at the *DefineEmployee* method in code\Employee class_v4.cs
- And to the *main* method too ;)

- The *DefineEmployee* method is public:
  - so I can call it from **outside** its class!
  - Here I call it from the *main*.

Break

- Questions so far?

Constructors

- **Remember the life cycle of an object?**
- When you create an object using *new*, C# actually tries to use a constructor (i.e. a special kind of method) to define the initial values of the attributes of the object.
  - if there is no constructor in a class (as in Employee),
    C# does some minimal work and leaves newly create the object as it is
- But we can write a **constructur** for a class, and then C# will use it

The book explains it like this:

<<The class constructor method is named the same as the class.
When an object instance of a class is instantiated by client code, the constructor method is executed.>>

Constructor for Employee

- In our case, the code inside the method DefineEmployee could be reused for the constructor

- Look at: code\Employee class_v5.cs
  to see how the constructor can be implemented
- A constructor has the same name as the class
- It looks like a method but…
  - Does not have a return type

Overloading methods

- Overloading methods is a useful/typical feature of OOP languages
- You overload methods in a class by **defining multiple methods** that have the **same name** but contain **different signatures**.
  - A method signature is a **combination of the name of the method and its parameter type list**
    void m(int a,int b) has a different signature than **void m(int a)** and of **void m(int a,string b)**
  - If you change the parameter type list, you create a different method signature

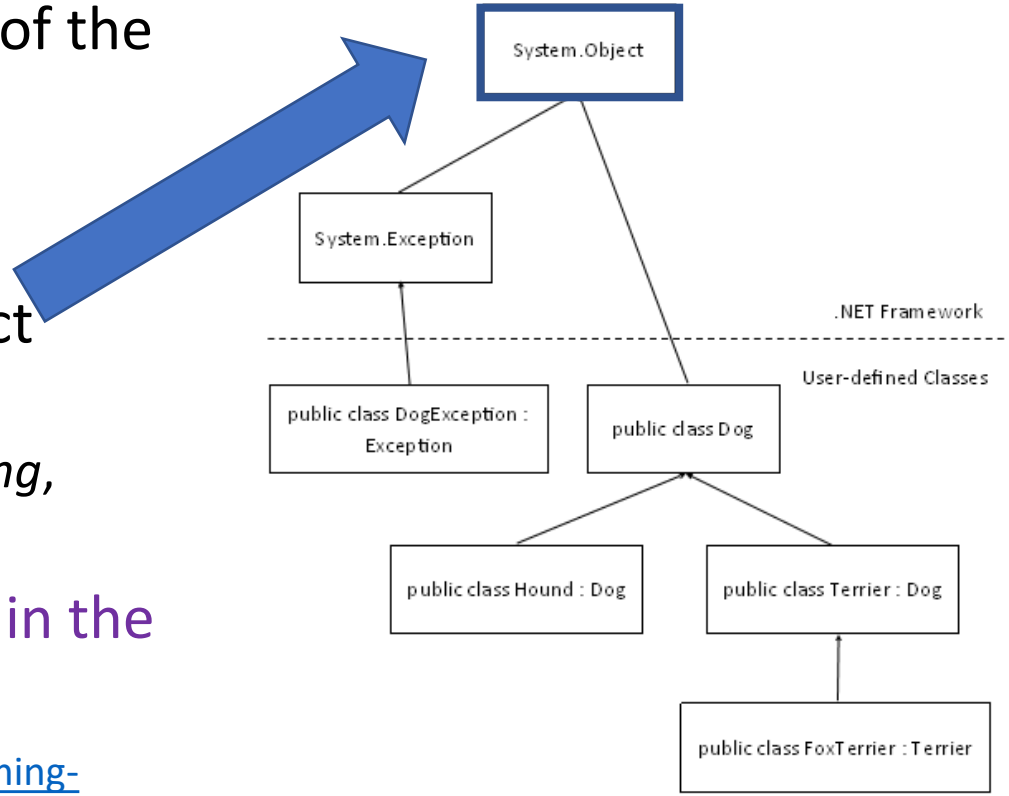
- **PRO:**
  the C# compiler will determine which method to execute by examining the parameter type list passed in by the client

- **Look at: code\Employee class_v6.cs to see how the DefineEmployee method can be overloaded, and how the *main* uses the new version of the method**
- Note: a **constructor** can also be **overloaded**! (see the code ;))

Overriding the ToString method

- All classes in C# derive from the **Object class**: *top* of the class hierarchy
  - https://learn.microsoft.com/en-us/dotnet/api/system.object?view=net-8.0
- Even a class like **Employee** has a superclass: Object
  - Some methods are inherited from Object, like *ToString*
  - This means: any class can **redefine its version** of *ToString*, i.e. *override the ToString method*
- More about overriding and the C# class hierarchy in the next lecture…

[read more here https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-override-the-tostring-method]

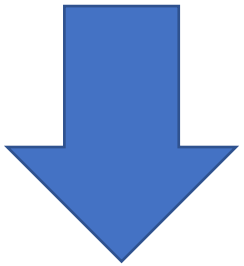


ToString overridden

- The advantage of overriding ToString is that
  - When you Console.Write an object, C# will try to use the ToString method
  - If there is no such method redefined in your class,
    the standard one from class Object is used -> **it only prints your class name**

- The signature of the method that overrides, must be exactly the same as the original method, including public/private
- And you must use the override keyword (to tell the compiler!):

```
public override string ToString(){  /* your code */ }
```



ToString overridden (cont.)

- Look in code\Employee class_v7.cs to see the ToString method + the main

```
public override string ToString() {
    return "Employee " + Name + ", Dept: " + Department;
}
```

```
static void Main(string[] args)
{
    Employee emp = new Employee(3,"abc", "XXX123!", "SillyWalks", "Andrea");
    Console.WriteLine(emp);
    Console.WriteLine("The ID of this employee is " + emp.EmployeeID);

    Console.WriteLine("------------------------------");
    Employee anotherEmp = new Employee();
    anotherEmp.DefineEmployee("Used Cars", "Bob");
    Console.WriteLine(anotherEmp);
}
```

Clean up your code

- Why having a ”default” Set and Get blocks for a private property?
  - Perhaps it is better to leave the attribute *public*
  - Moreover from C# ver 6 we can write:

```
public string Address { get; set; } = "Someplace 1,Kolding";
```

AKA auto-implemented properties

- Look at: code\Employee class_final.cs

Usage in the main ...

```
Console.WriteLine(emp.Address);
emp.Address = "SomeOtherPlace";
Console.WriteLine(emp.Address);
```

Break

- Questions so far?

Tasks for next time

(but you can start here, in groups if you like)

Products in a shopping cart

Draw a class diagram for the Product class

- You are asked to implement a program that can:
  - Define a few products; every product has a name, a product code and a price
  - Add them to a shopping cart
  - Calculate and print the total cost of the shopping cart
- You will need a class Product with attributes and methods (and remember *encapsulation*!)
- a main method where you can create, initialize and add the products to the shopping cart, and where you will calculate and print the total cost,
- And you can use a *List<Product>* to implement the shopping cart
- Suggestion: override the ToString for the Product class to help you ”see” what your products

Example

Product ”soap”, code 123, price 15 kr

Product ”gouda cheese” , code 554, price 70,95 kr

Total should be: 15+70,95 => so 85,95 kr

- Test your program with at least 3 products in the shopping cart.
- **[CHALLENGE]** Add some code to your main, so that you can find the cheapest product, and print its name
  - Here it would be ”soap”
- Finally: could you turn the List shopping chart into an object of a new class ShoppingCart, with the calculation method?