Programmering og Problemløsning

6 December 2016

Christina Lioma

c.lioma@di.ku.dk

Today's lecture

- Encapsulation
  - Data hiding
  - Access modifiers
  - Instance and Static members

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

Hi, I'm Bob

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

- Class definition

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

- Class definition
- Class declaration & class primary constructor

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

- Class definition
- Class declaration & class primary constructor
- Instantiate new object

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

- Class definition
- Class declaration & class primary constructor
- Instantiate new object
- Use instantiated object

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

- Class definition
- Class declaration & class primary constructor
- Instantiate new object
- Use instantiated object
- Class inference

```
open System
type Robot(name ~~: string~~) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

- Class definition
- Class declaration & class primary constructor
- Instantiate new object
- Use instantiated object
- Class inference
- Type inference

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

The class is called **Robot**

The object instance is called **bob**

Object instance bob has an attribute Name whose value is **Bob**

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

- What if we wish to change the value of bob's name?

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

- What if we wish to change the value of bob's name?

We pass the new value of name as an argument to Robot()

```
open System
type Robot(name : string) = class
   member x.Name = name
   member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
```

- What if we wish to start with “Bob” (because it is friendly) and then change it to “Robert” (more serious)?

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

- This does not work. **“Property 'Name' is not readable”.** Why?

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot("Bob")
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

- This does not work. **“Property 'Name' is not readable”.**

  name is immutable

```
open System
type Robot() = class
    let mutable name = "Bob"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot()
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

- Let’s make name mutable

```
open System
type Robot() = class
    let mutable name = "Bob"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot()
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

- Still does not work, even though name is now mutable. **“Property 'Name' cannot be set”.** Why?

```
open System
type Robot() = class
    let mutable name = "Bob"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot()
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

- Still does not work, even though name is now mutable. **“Property 'Name' cannot be set”.** Why?
- Why did it work in bankExample.fs?

What is the difference?

Robot object

Account object

What is the difference?

Robot object
name mutable

Account object
amount mutable

What is the difference?

Robot object

name mutable

Cannot change name in object instance

Account object

amount mutable

Can change amount in object instance

What is the difference?

Robot object

name mutable

Cannot change name in object instance

bob.Name **<-** “Robert” (direct assignment)

Account object

amount mutable

Can change amount in object instance

homer.**Withdraw** 50 (assignment via class method)

Encapsulation & data hiding

Inside the class: all members are accessible

Outside the class: class attributes are only accessible through the class methods

Encapsulation & data hiding

Inside the class: all members are accessible

Outside the class: class attributes are only accessible through the class methods

```
type Robot() = class
   let mutable name = "Bob"
   member x.Name = name
   member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot()
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

“Property 'Name' cannot be set”

Encapsulation & data hiding

Inside the class: all members are accessible

Outside the class: class attributes are only accessible though the class methods

```
type Robot() = class
   let mutable name = "Bob"
   member x.Name = name
   member x.SayHello() = printfn "Hi, I'm %s" x.Name
   member x.Rename(value) = name <- value
end
let bob = new Robot()
bob.SayHello()                                  Hi, I'm Bob
bob.Rename("Robert")                            Hi, I'm Robert
bob.SayHello()
```

When we define an abstract object in a class

- Glue together its attributes & methods into a single entity

When we define an abstract object in a class

- Glue together its attributes & methods into a single entity
- We hide its attributes from the outside (of the class)

When we define an abstract object in a class

- Glue together its attributes & methods into a single entity
- We hide its attributes from the outside (of the class)
- Access attributes from outside:
  - specify an appropriate class method for accessing them

When we define an abstract object in a class

- Glue together its attributes & methods into a single entity
- We hide its attributes from the outside (of the class)
- Access attributes from outside:
  - specify an appropriate class method for accessing them

OR

  - define attributes to be **outside-accessible** without a class method

Accessing class attributes without class methods

Access attribute: read and/or write to it

Accessing class attributes without class methods

Access attribute: read and/or write to it

Outside-accessible attribute: can read and/or write to it from **outside** the class (without class methods)

Accessing class attributes without class methods

Access attribute: read and/or write to it

Outside-accessible attribute: can read and/or write to it from **outside** the class (without class methods)

Must be specified in the attribute definition **inside** the class (part of the template)

Accessing class attributes without class methods

Access attribute: read and/or write to it

Outside-accessible attribute: can read and/or write to it from **outside** the class (without class methods)

Must be specified in the attribute definition **inside** the class (part of the template)

Specify with get() and set() permissions

(special methods for class attributes)

Accessing class attributes without class methods

Access attribute: read and/or write to it

Outside-accessible attribute: can read and/or write to it from **outside** the class (without class methods)

Must be specified in the attribute definition **inside** the class (part of the template)

Specify with get() and set() permissions

(special methods for class attributes):

- get() allows reading the value of a class attribute
- set() allows setting a new value to a class attribute (only for mutable attributes)

get() and set() syntax

Without get() and set()

member alias.AttributeName = current-value

With get() and set()

member alias.AttributeName

    with get() = current-value

    and set(new-value) = *some-assignment*

```
open System
type Robot() = class
    let mutable name = "Bob"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot()
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

“Property 'Name' cannot be set”

```
open System
type Robot() = class
    let mutable name = "Bob"
    member x.Name
            with get() = name
            and set(value) = name <- value
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot()
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

```
open System
type Robot() = class
   let mutable name = "Bob"
   member x.Name
         with get() = name
         and set(value) = name <- value
   member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot()
bob.SayHello()
bob.Name <- “Robert”
bob.SayHello()
```

Hi, I’m Bob

Hi, I’m Robert

Without get() and set(), class members are hidden & protected

Without get() and set(), class members are hidden & protected

With get() and set(), class members become:

- Visible outside the class
- Modifiable outside the class (less protected)

Use get() & set() to sanitise input

```
type Robot() = class
   let mutable name = "Bob"
   member x.Name
          with get() = name
          and set(value) =
                if name = “idiot” then
                      raise (new Exception(“Cannot do this!”))
                else
                      name <- value
   member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let bob = new Robot()
bob.SayHello()
bob.Name <- “idiot”
bob.SayHello()
```

get() and set() alternative syntax

```
member x.Name
        with get() = name
        and set(value) = name <- value
```

OR

```
member x.Name with get() = name
member x.Name with set(value) = name <- value
```

Groups of 2 people (5 minutes)

Protect the earth from falling meteors by shooting them with lasers

- Lasers have initial power of 50 units each
- Their power is consumed every time they find a target (-1) or are fired (-10)
- Lasers respond to: find target, shoot
- Lasers do not work without power
- We should be able to recharge lasers without using a class method

One *very* basic way of doing this…

Object

- Attributes

- Methods

One *very* basic way of doing this…

Object: **Laser**

- Attributes:
  - **name**
  - **power**
- Methods:
  - **find target**
  - **shoot**

One *very* basic way of doing this…

Object: Laser

- Attributes:
  - name (**immutable**)
  - power (**mutable**)
- Methods:
  - find target
  - shoot

One *very* basic way of doing this…

Object: Laser

- Attributes:
  - name (immutable)
  - power (mutable, **accessible outside class**)
- Methods:
  - find target
  - shoot

One *very* basic way of doing this…

Object: Laser

- Attributes:
  - name (immutable)
  - power (mutable, accessible outside class, **if no power**)
- Methods:
  - find target
  - shoot

```
type Laser(name) = class
   let mutable power = 50
   member x.Name = name
   member x.Power
      with get() = power and set(value) =
         if  value < 1 then raise (new Exception("Laser out of power."))
         else power <- value
   member x.FindTarget() = power <- power - 1
   member x.Shoot() = power <- power - 10
end
let laser1 = new Laser("Laser-1")
laser1.FindTarget()
laser1.Shoot()
printfn "%s has %i power units left" laser1.Name laser1.Power
laser1.Power <- 50
printfn "%s has %i power units left" laser1.Name laser1.Power
```

Use Laser class to create many laser instances

Use Laser class to create many laser instances

Each laser instance can find & shoot different targets

Each laser instance can have different power left

Use Laser class to create many laser instances

Each laser instance can find & shoot different targets

Each laser instance can have different power left

- Laser-1 can have 39 units of power
- Laser-2 can have 17 units of power

The values of each object instance are stored separately in memory

Use Laser class to create many laser instances

Each laser instance can find & shoot different targets

Each laser instance can have different power left

- Laser-1 can have 39 units of power
- Laser-2 can have 17 units of power

The values of each object instance are stored separately in memory

power is an instance member of class Laser

Use Laser class to create many laser instances

Each laser instance can find & shoot different targets

Each laser instance can have different power left

- Laser-1 can have 39 units of power
- Laser-2 can have 17 units of power

The values of each object instance are stored separately in memory

power is an instance member of class Laser

Instance members: only apply to object instances

Use Laser class to create many laser instances

Each laser instance can find & shoot different targets

Each laser instance can have different power left

- Laser-1 can have 39 units of power
- Laser-2 can have 17 units of power

The values of each object instance are stored separately in memory

power is an instance member of class Laser

Instance members: only apply to object instances

Static members: apply to the whole class

class Laser:

name, power

class Laser:

name, power

- Each Laser instance has its own values of name & power, stored in different memory locations. Each of these values is associated to a different instance. **Instance Members**

class Laser:

serialNo, name, power

- Each Laser instance has its own values of name & power, stored in different memory locations. Each of these values is associated to a different instance. **Instance Members**
- serialNo has different value per instance. **Instance Member**

class Laser:

serialNo, name, power

- Each Laser instance has its own values of name & power, stored in different memory locations. Each of these values is associated to a different instance. **Instance Members**
- serialNo has different value per instance. **Instance Member**
- To assign serialNo, need to know total number of lasers created

class Laser:

totalNo, serialNo, name, power

- Each Laser instance has its own values of name & power, stored in different memory locations. Each of these values is associated to a different instance. **Instance Members**
- serialNo has different value per instance. **Instance Member**
- To assign serialNo, need to know total number of lasers created
- totalNo has same value in all instances. **Static Member**

class Laser:

totalNo, serialNo, name, power

- Each Laser instance has its own values of name & power, stored in different memory locations. Each of these values is associated to a different instance. **Instance Members**
- serialNo has different value per instance. **Instance Member**
- To assign serialNo, need to know total number of lasers created
- totalNo has same value in all instances. **Static Member**

Static does not mean immutable!

Instance & Static syntax

Default syntax creates instance members

Different syntax for static members

```
type SomeClass(property : int) = class
    member x.Property = property
    static member StaticProperty = “This is a static property”
    …
end
```

Instance & Static syntax

Default syntax creates instance members

Different syntax for static members

```
type SomeClass(property : int) = class
    member x.Property = property
    static member StaticProperty = "This is a static property"
    …
end
```

No self-identifier. Why?

Instance & Static syntax

Default syntax creates instance members

Different syntax for static members

```
type SomeClass(property : int) = class
    member x.Property = property
    static member StaticProperty = "This is a static property"
    …
end
```

No self-identifier. Why?

Because no object instance currently in scope

Valid for all object instances of this class

Recap today's lecture

- Data hiding
- Access modifiers
- Instance and Static members