Programmering og Problemløsning

5 December 2019

Christina Lioma

c.lioma@di.ku.dk

Today's lecture

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

```
type Class()
    attribute
    method()

let myInstance = new Class()
myInstance.method()
```

```
type Class()
    attribute
    method()
```

Class declaration & class constructor

Class members

```
let myInstance = new Class()
myInstance.method()
```

Make object instance

Use object instance

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot("Max")
robot1.SayHello()
```

Hi, I'm Max

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot("Max")
robot1.SayHello()
```

The class is called **Robot**

The object instance is called **robot1**

Object instance robot1 has an attribute Name whose value is **Max**

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot("Max")
robot1.SayHello()
```

- How can we change the value of robot1'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 robot1 = new Robot("Max")
robot1.SayHello()
```

- How can we change the value of robot1'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 robot1 = new Robot("Max")
robot1.SayHello()
```

- How can we start with “Max” and then change it to “Maxwell”?

```
open System
type Robot(name : string) = class
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot("Max")
robot1.SayHello()
```

- name is immutable

```
open System
type Robot() = class
    let mutable name = "Max"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot()
robot1.SayHello()
```

- Let’s make name mutable

```
open System
type Robot() = class
    let mutable name = "Max"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot()
robot1.SayHello()
```

- Now name is mutable
- How can we start with “Max” and then change it to “Maxwell”?

```
open System
type Robot() = class
    let mutable name = "Max"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot()
robot1.SayHello()
robot1.Name <- "Maxwell"
robot1.SayHello()
```

```
open System
type Robot() = class
    let mutable name = "Max"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot()
robot1.SayHello()
robot1.Name <- "Maxwell"
robot1.SayHello()
```

- This does not work. **“Property 'Name’ cannot be set”.** Why?

Robot object

Account object

Robot object
name mutable

Account object
amount mutable

Robot object

name mutable

Cannot change name in object instance

Account object

amount mutable

Can change amount in object instance

Robot object

name mutable

Cannot change name in object instance

robot1.Name **<-** “Maxwell” (direct assignment)

Account object

amount mutable

Can change amount in object instance

max.**Deposit(**100**)** (assignment via class method)

Encapsulation & data hiding

Inside the class: all members are accessible

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

Encapsulation & data hiding

Inside the class: all members are accessible

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

```
type Robot() = class
    let mutable name = "Max"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot()
robot1.SayHello()
robot1.Name <- “Maxwell”
robot1.SayHello()
```

“Property 'Name' cannot be set”

Encapsulation & data hiding

Inside the class: all members are accessible

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

```
type Robot() = class
   let mutable name = "Max"
   member x.Name = name
   member x.SayHello() = printfn "Hi, I'm %s" x.Name
   member x.Rename(value) = name <- value
end
let robot1 = new Robot()
robot1.SayHello()
robot1.Rename("Maxwell")
robot1.SayHello()
```

Hi, I'm Max
Hi, I'm Maxwell

Data hiding

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

Data hiding

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

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

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

Specify with get() and set(): special methods for 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 x.AttributeName = current-value
```

With get() and set()

```
member x.AttributeName
    with get() = current-value
    and set(new-value) = some-assignment
```

```
open System
type Robot() = class
    let mutable name = "Max"
    member x.Name = name
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot()
robot1.SayHello()
robot1.Name <- “Maxwell”
robot1.SayHello()
```

“Property 'Name' cannot be set”

```
open System
type Robot() = class
    let mutable name = "Max"
    member x.Name
            with get() = name
            and set(value) = name <- value
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot()
robot1.SayHello()
robot1.Name <- “Maxwell”
robot1.SayHello()
```

```
open System
type Robot() = class
    let mutable name = "Max"
    member x.Name
            with get() = name
            and set(value) = name <- value
    member x.SayHello() = printfn "Hi, I'm %s" x.Name
end
let robot1 = new Robot()
robot1.SayHello()
robot1.Name <- “Maxwell”
robot1.SayHello()
```

Hi, I’m Max
Hi, I’m Maxwell

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 = "Max"
   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 robot1 = new Robot()
robot1.SayHello()
robot1.Name <- “idiot”
```

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 meteors by shooting them with lasers. Write a Laser class that implements the following:

- Lasers have initial power of 50 units each
- Their power is consumed every time they find a target (-1) or shoot (-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

Groups of 2 people (5 minutes)

Protect the earth from meteors by shooting them with lasers. Write a Laser class that implements the following:

- Lasers have initial power of 50 units each
- Their power is consumed every time they find a target (-1) or shoot (-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

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

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
```

```
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
```

Recap

Class defines two major aspects of an instance:

- The attributes that are used in each instance

- The operations that are performed on each instance

An instance is related to the class from which it was created (**“instance-of” relationship**)

Recap

Class defines two major aspects of an instance:

- The attributes that ~~are~~ **can be** used in each instance

- The operations that ~~are~~ **can be** performed on each instance

An instance is related to the class from which it was created (“instance-of” relationship)

Recap

Class defines two major aspects of an instance:

- The attributes that ~~are~~ can be used in each instance **but not their values**
- The operations that ~~are~~ can be performed on each instance

An instance is related to the class from which it was created (“instance-of” relationship)

Use Laser class to create many laser instances

Use Laser class to create many laser instances

Each laser instance can find & shoot different targets

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

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: their value applies 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: their value applies to object instances

Static members: their value applies 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

```
type SomeClass(property : int) = class
    member x.Property = 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
```

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 in scope

Valid for all object instances of this class

Class

Laser()

member name

member count

member showCount()

Class

Laser()

member name

member count

member showCount()

Object instantiation

laser1 = new Laser(..)

Class

Laser()

member name

member count

member showCount()

Object instantiation

laser1 = new Laser(..)

name=SuperLaser

count=1

…

…

Class

Laser()

member name

static member count

member showCount()

Object instantiation

laser1 = new Laser(..)

name=SuperLaser

count=~~1~~ 2

…

laser2 = new Laser(..)

name=GigaLaser

count=2

Class

Laser()

member name

static member count

member showCount()

Object instantiation

laser1 = new Laser(..)

name=SuperLaser

count=~~1 2~~ 3

…

laser2 = new Laser(..)

name=GigaLaser

count=~~2~~ 3

…

laser3 = new Laser(..)

name=TurboLaser

count=3

…

Class

Laser()

instance member name

static member count

member showCount()

Object instantiation

laser1 = new Laser(..)

name=SuperLaser

count=~~1 2~~ 3

…

laser2 = new Laser(..)

name=GigaLaser

count=~~2~~ 3

…

laser3 = new Laser(..)

name=TurboLaser

count=3

…

Class

Laser()

instance member name

static member count

? member showCount()

Must I use a static method to access a static attribute?

Object instantiation

laser1 = new Laser(..)

name=SuperLaser

count=~~1 2~~ 3

…

laser2 = new Laser(..)

name=GigaLaser

count=~~2~~ 3

…

laser3 = new Laser(..)

name=TurboLaser

count=3

…

Class

Laser()

instance member name

static member count

? member showCount()

Must I use a static method to access a static attribute?

1. ***A static method can access a static attribute***
2. ***An instance method can access a static attribute***

Object instantiation

laser1 = new Laser(..)

name=SuperLaser

count=~~1~~ ~~2~~ 3

…

laser2 = new Laser(..)

name=GigaLaser

count=~~2~~ 3

…

laser3 = new Laser(..)

name=TurboLaser

count=3

…

Class

Laser()

instance member name

static member count

? member showCount()

Must I use a static method to access a static attribute?

1. ***A static method can access a static attribute***
2. ***An instance method can access a static attribute***

It depends on HOW I want to access the static attribute

Object instantiation

laser1 = new Laser(..)

name=SuperLaser

count=~~1 2~~ 3

…

laser2 = new Laser(..)

name=GigaLaser

count=~~2~~ 3

…

laser3 = new Laser(..)

name=TurboLaser

count=3

…

```
type Laser(name) =
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
```

```
type Laser(name) =
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =
        printfn ”Lasers created: %i” Laser.Count
```

```
type Laser(name) =
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =
        printfn ”Lasers created: %i” Laser.Count
```

Instance method to access static attribute

```
type Laser(name) =
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =
        printfn ”Lasers created: %i” Laser.Count
```

Static method to access static attribute

```
type Laser(name) =
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =
        printfn ”Lasers created: %i” Laser.Count
let laser1 = new Laser(“Super Laser”)
laser1.ShowCountI()
Laser.ShowCountS()
```

Lasers created: 1
Lasers created: 1

```
type Laser(name) =
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =
        printfn ”Lasers created: %i” Laser.Count
```

~~let laser1 = new Laser(“Super Laser”)~~

laser1.ShowCountI() → ***Does not work***

Laser.ShowCountS() → ***Works***

```
type Laser(name) =
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =
        printfn ”Lasers created: %i” Laser.Count
```

~~let laser1 = new Laser(“Super Laser”)~~

laser1.ShowCountI() → ***Does not work***

Laser.ShowCountS() → ***Works***

Output?

```
type Laser(name) =
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =
        printfn ”Lasers created: %i” Laser.Count
```

~~let laser1 = new Laser(“Super Laser”)~~

laser1.ShowCountI() → ***Does not work***

Laser.ShowCountS() → ***Works***

Lasers created: 0

```
type Laser(name) =
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =
        printfn ”Lasers created: %i” Laser.Count
let laser1 = new Laser(“Super Laser”)
laser1.ShowCountI()
Laser.ShowCountS()
```

Flow of execution

→ ***Does not work***

→ ***Works*** ***1***

Lasers created: 0

```
type Laser(name) =                                    2
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =
        printfn ”Lasers created: %i” Laser.Count
let laser1 = new Laser(“Super Laser”)
laser1.ShowCountI()     → Does not work
Laser.ShowCountS()      → Works          1
```

Flow of execution

Lasers created: 0

```
type Laser(name) =                                  2
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count
    member x.ShowCountI() = printfn "Lasers created: %i" Laser.Count
    static member ShowCountS() =                    3
        printfn "Lasers created: %i" Laser.Count
let laser1 = new Laser("Super Laser")
laser1.ShowCountI()     → Does not work
Laser.ShowCountS()      → Works                     1
```

Flow of execution

Lasers created: 0

```
type Laser(name) =                                  2
    static let mutable count = 0
    do count <- count + 1
    member x.Name = name
    static member Count = count                     4
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =                    3
        printfn ”Lasers created: %i” Laser.Count
let laser1 = new Laser(“Super Laser”)
laser1.ShowCountI()     → Does not work
Laser.ShowCountS()      → Works                     1
```

Flow of execution

Lasers created: 0

```
type Laser(name) =                                  2
    static let mutable count = 0                    5
    do count <- count + 1
    member x.Name = name
    static member Count = count                     4
    member x.ShowCountI() = printfn ”Lasers created: %i” Laser.Count
    static member ShowCountS() =                    3
        printfn ”Lasers created: %i” Laser.Count
let laser1 = new Laser(“Super Laser”)
laser1.ShowCountI()     → Does not work
Laser.ShowCountS()      → Works                     1
```

Flow of execution

Lasers created: 0

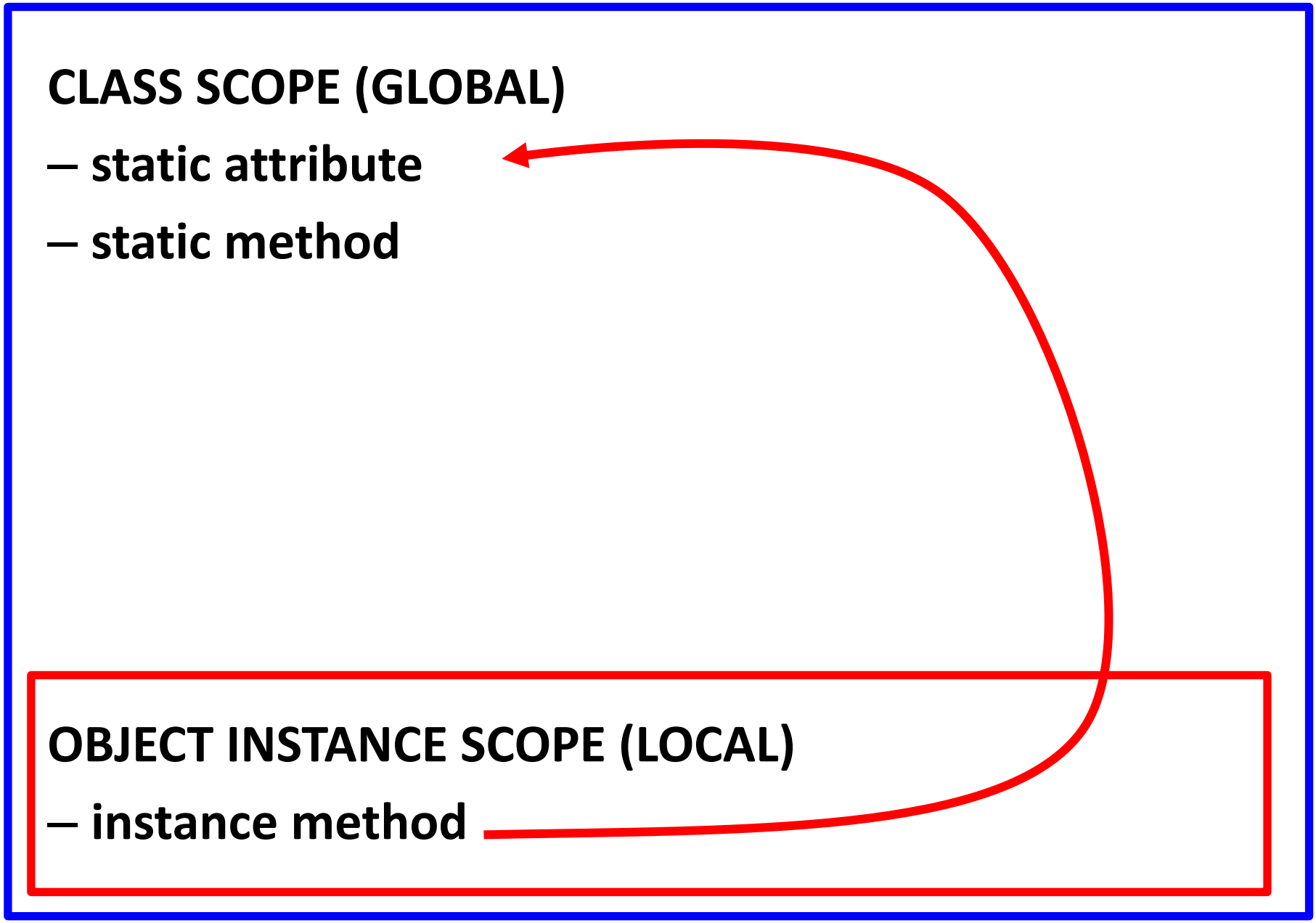
CLASS SCOPE (GLOBAL)

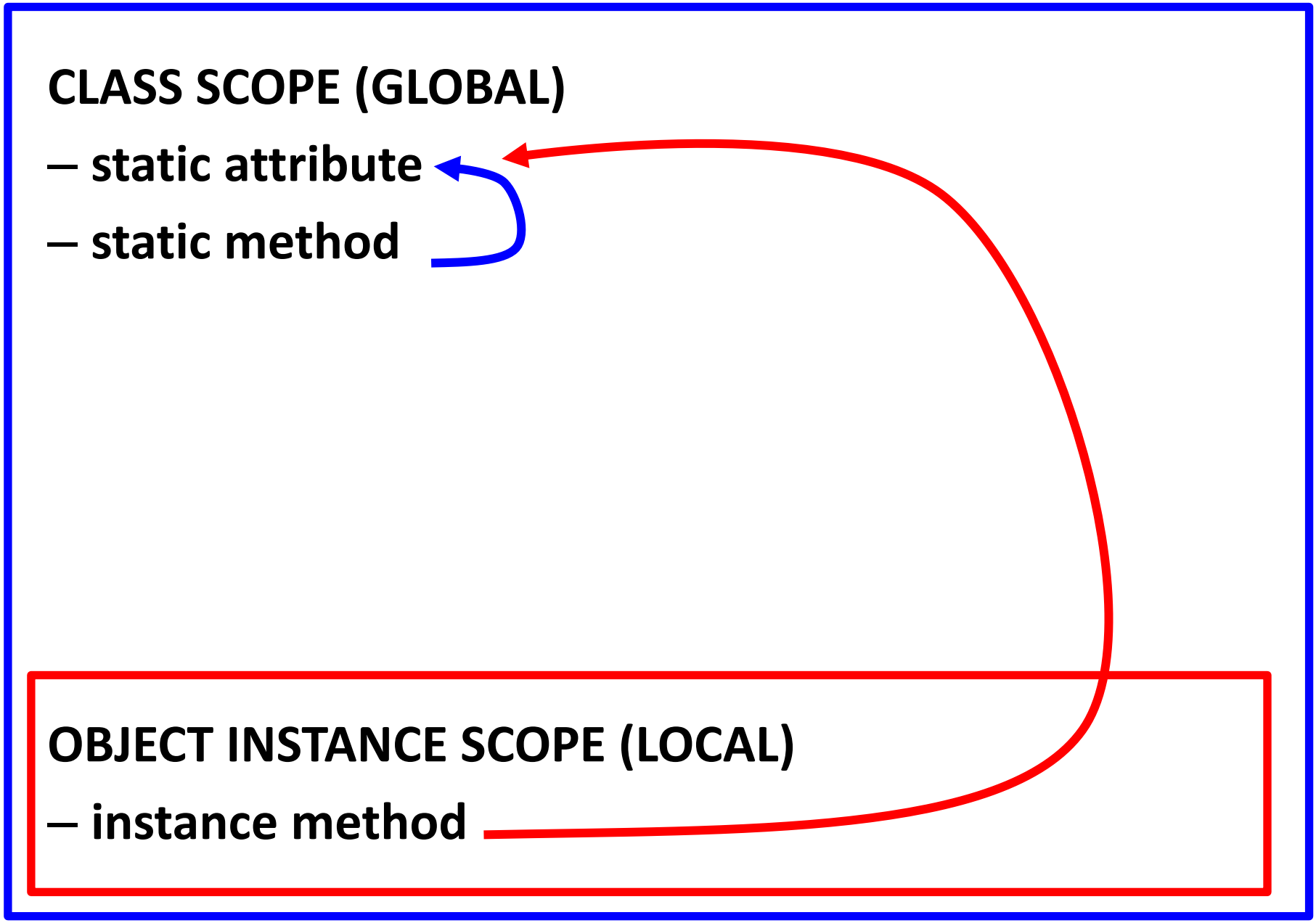
– static attribute

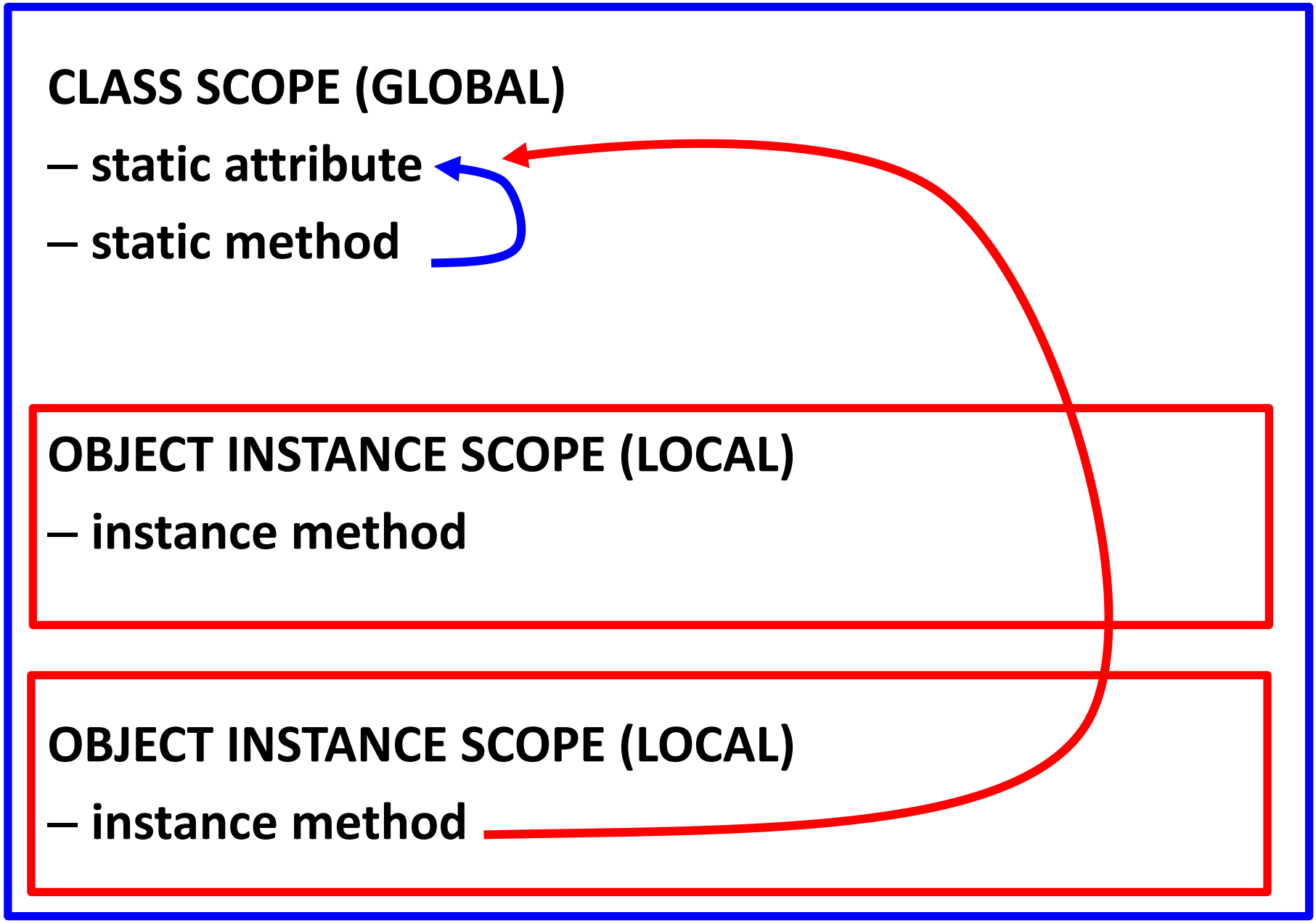
– static method

OBJECT INSTANCE SCOPE (LOCAL)

– instance method


CLASS SCOPE (GLOBAL)
– static attribute
– static method
OBJECT INSTANCE SCOPE (LOCAL)
– instance method


CLASS SCOPE (GLOBAL)
– static attribute
– static method
OBJECT INSTANCE SCOPE (LOCAL)
– instance method


CLASS SCOPE (GLOBAL)
– static attribute
– static method
OBJECT INSTANCE SCOPE (LOCAL)
– instance method
OBJECT INSTANCE SCOPE (LOCAL)
– instance method

Recap today's lecture

- Data hiding
- Access modifiers
- Instance and Static members