Programmering og Problemløsning

11 December 2018

Christina Lioma

c.lioma@di.ku.dk

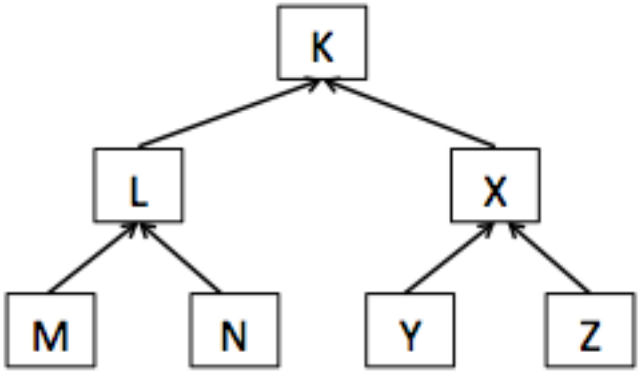
Today's lecture

Class Inheritance

- What can be inherited (and what cannot)
- Overriding
- Definition & Implementation
- Overshadowing

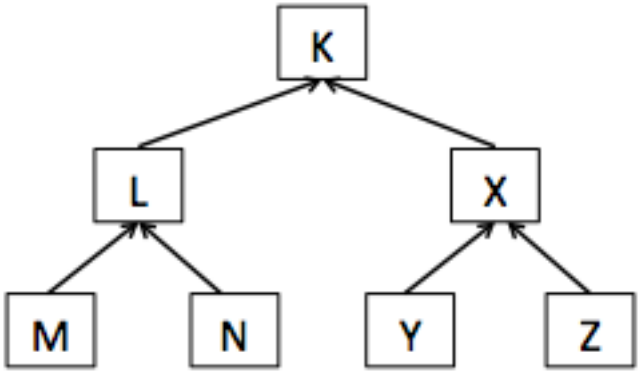
Inheritance

- In F#, *Derived* has only one direct *Base*
- *Derived* inherits all members from *Base*
- If *Base* has additional constructors, the constructor(s) to be inherited must be specified



Inheritance

- In F#, *Derived* has only one direct *Base*
- *Derived* inherits all **non-private** members from *Base*
- If *Base* has additional constructors, the constructor(s) to be inherited must be specified



```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
```

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
type SpeedLaser(p, a) =
    inherit Laser(p, a)
```

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
laser1.Shoot()
```

Power left: 79.000000

```
type Laser(p, a) =
        let mutable power = p
        let mutable accuracy = a
        member private x.Shoot() =
                power <- power – 1.0
                printfn “Power left: %f” power
type SpeedLaser(p, a) =
        inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
laser1.Shoot()
```

Make Shoot() private

Error: ‘Shoot’ is not accessible

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member private x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
laser1.Shoot()
```

This compiles.

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member private x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
laser1.Shoot()
```

This compiles.
What is inherited?

```
type Laser(p, a) =
    do printfn "TEST"
    let mutable power = p
    let mutable accuracy = a
    member private x.Shoot() =
        power <- power – 1.0
        printfn "Power left: %f" power
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
```

TEST

```
type Laser(p, a) =
        do printfn "TEST"
        let mutable power = p
        let mutable accuracy = a
        member private x.Shoot() =
                power <- power – 1.0
                printfn "Power left: %f" power
type SpeedLaser(p, a) =
        inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
```

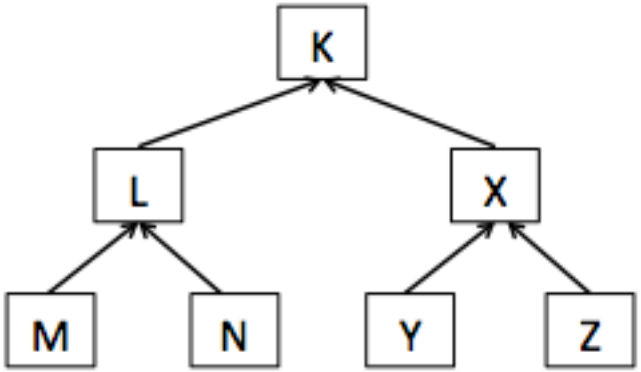
INHERITED

NOT INHERITED

TEST

The derived class inherits from its base class:

- All non-private members
- Additional constructor(s) (must be specified)
- The primary constructor
- do-bindings in the constructor body



```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
    new(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        Laser(floatP, floatA)
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80, 90)
laser1.Shoot()
```

```
type Laser private (p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
    new(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        Laser(floatP, floatA)
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80, 90)
laser1.Shoot()
```

What if the Base primary constructor is private?

```
type Laser private (p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
    new(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        Laser(floatP, floatA)
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80, 90)
laser1.Shoot()
```

What if the Base primary constructor is private?

It works. Why?

```
type Laser private (p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
    new(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        Laser(floatP, floatA)
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80, 90)
laser1.Shoot()
```

What if the Base primary constructor is private?

It works. Why? Because it resolves the method overload

```
type Laser private (p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
    new(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        Laser(floatP, floatA)
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80, 90)
laser1.Shoot()
```

What if the Base primary constructor is private?

It works. Why? Because it resolves the method overload

Output: float or integer?

```
type Laser private (p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
    new(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        Laser(floatP, floatA)
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80, 90)
laser1.Shoot()
```

Power left: 79.000000

What if the Base primary constructor is private?

It works. Why? Because it resolves the method overload

Output: float ~~or integer?~~

```
type Laser private (p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
    new(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        Laser(floatP, floatA)
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
laser1.Shoot()
```

What if the Base primary constructor is private?

What about this?

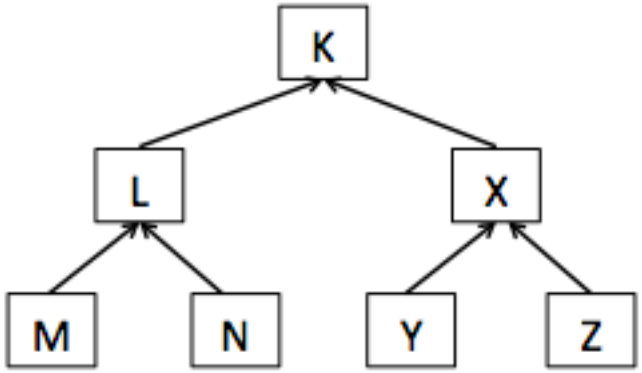
```
type Laser private (p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power
    new(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        Laser(floatP, floatA)
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
laser1.Shoot()
```

What if the Base primary constructor is private?

What about this? Does not work because input arguments can only be integers (we have not inherited the method overload)

The derived class inherits from its base class:

- All non-private members
- Non-private constructor(s) (must be specified)
- do-bindings in the non-private constructor body



```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()

let laser1 = Laser()
laser1.ShowID()                                    Galaxy235
```

```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()

let laser1 = Laser()
laser1.ShowID()                    Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                    Galaxy235
```

```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    member x.Power = 70
    member x.ShowPower() = System.Console.Write(x.Power)
let laser1 = Laser()
laser1.ShowID()                                    Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                                    Galaxy235
```

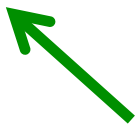
```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    member x.Power = 70
    member x.ShowPower() = System.Console.Write(x.Power)
let laser1 = Laser()
laser1.ShowID()                          Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                          Galaxy235
laser2.ShowPower()                       70
```

```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    member x.Power = 70
    member x.ShowPower() = System.Console.Write(x.Power)
let laser1 = Laser()
laser1.ShowID()                    Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                    Galaxy235
laser2.ShowPower()                 70
laser1.ShowPower()                 what does this output?
```

```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    member x.Power = 70
    member x.ShowPower() = System.Console.Write(x.Power)
let laser1 = Laser()
laser1.ShowID()                    Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                    Galaxy235
laser2.ShowPower()                 70
laser1.ShowPower()                 “ShowPower is not defined”
```

```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```



I WANT TO OVERRIDE THIS METHOD IN THE DERIVED CLASS

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
```

Three steps to override an inherited member:

Three steps to override an inherited member:

- State in the base class that the member can be overridden

Three steps to override an inherited member:

- State in the base class that the member can be overridden
- State in the base class how the member works if it is not overridden

Three steps to override an inherited member:

- State in the base class that the member can be overridden
- State in the base class how the member works if it is not overridden
- State in the derived class how the member is overridden

Three steps to override an inherited member:

- State in the base class that the member can be overridden

  use keyword *abstract*
- State in the base class how the member works if it is not overridden

- State in the derived class how the member is overridden

Three steps to override an inherited member:

- State in the base class that the member can be overridden

  use keyword *abstract*
- State in the base class how the member works if it is not overridden

  use keyword *default*
- State in the derived class how the member is overridden

Three steps to override an inherited member:

- State in the base class that the member can be overridden

  use keyword *abstract*
- State in the base class how the member works if it is not overridden

  use keyword *default*
- State in the derived class how the member is overridden

  use keyword *override*

```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

```
type Laser() =
    member x.ID = “Galaxy235”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

CAN BE OVERRIDEN

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

CAN BE OVERRIDEN

IF NOT OVERRIDEN, USE THIS

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

DEFINITION

IMPLEMENTATION

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

DEFINITION

DEFINITION SYNTAX:

abstract member MemberName **:** data type

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

DEFINITION

DEFINITION SYNTAX:

abstract member MemberName **:** data type

unit data type: indicates the absence of a value (placeholder when no value exists / is needed)
https://msdn.microsoft.com/en-us/library/dd483472.aspx

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

IMPLEMENTATION

DEFINITION SYNTAX:

abstract member MemberName **:** data type

IMPLEMENTATION SYNTAX:

default selfIdentifier**.**MemberName **=** …

unit data type: indicates the absence of a value (placeholder when no value exists / is needed)
https://msdn.microsoft.com/en-us/library/dd483472.aspx

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
```

DEFINITION

IMPLEMENTATION

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
```

DEFINITION

IMPLEMENTATION

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
```

DEFINITION

IMPLEMENTATION

NEW IMPLEMENTATION

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
```

“**override**” keyword: re-implements the method of the base class

“**base**” keyword: accesses directly members of the base class

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)

let laser1 = Laser()
laser1.ShowID()
let laser2 = SpeedLaser()
laser2.ShowID()
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)

let laser1 = Laser()
laser1.ShowID()                          Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                          Galaxy235.v2
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)

let laser1 = Laser()
laser1.ShowID()                                  Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                                  Galaxy235.v2
```

CAN OVERRIDE ATTRIBUTES TOO

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
    abstract member                    :
    default x.                         =
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
    abstract member                    :                    DEF
    default x.                         =                    IMPL
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
    abstract member CompatibleWith : string list
    default x.CompatibleWith = []
```

DEF

IMPL

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
    abstract member CompatibleWith : string list        DEF
    default x.CompatibleWith = []                       IMPL
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
    abstract member CompatibleWith : string list          DEF
    default x.CompatibleWith = []                         IMPL
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
    override x.CompatibleWith = [“Space0”; “Space9”]      IMPL
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
    abstract member CompatibleWith : string list          DEF
    default x.CompatibleWith = []                         IMPL
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
    override x.CompatibleWith = [“Space0”; “Space9”]      IMPL
let laser1 = Laser()
let laser2 = SpeedLaser()
System.Console.Write(laser1.CompatibleWith)
System.Console.Write(laser2.CompatibleWith)
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
    abstract member CompatibleWith : string list          DEF
    default x.CompatibleWith = []                         IMPL
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
    override x.CompatibleWith = [“Space0”; “Space9”]      IMPL
let laser1 = Laser()
let laser2 = SpeedLaser()
System.Console.Write(laser1.CompatibleWith)  []
System.Console.Write(laser2.CompatibleWith)  [Space0; Space9]
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
    override x.ShowID() = System.Console.Write(base.ID+”.v3”)
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
    override x.ShowID() = System.Console.Write(base.ID+”.v3”)

let laser2 = SpeedLaser()
laser2.ShowID()
```

What does this output?

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
    override x.ShowID() = System.Console.Write(base.ID+”.v3”)

let laser2 = SpeedLaser()
laser2.ShowID()
```

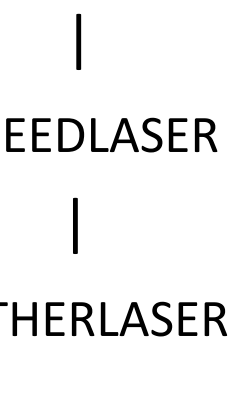
More than one override implements 'ShowID : unit -> unit'

Overriding in inheritance

- A *Base* class member can be overridden **only once** in **the same** *Derived* class

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
type OtherLaser() =
    inherit SpeedLaser()
    override x.ShowID() = System.Console.Write(base.ID+”.v3”)
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
type OtherLaser() =
    inherit SpeedLaser()
    override x.ShowID() = System.Console.Write(base.ID+”.v3”)
```



```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
type OtherLaser() =
    inherit SpeedLaser()
    override x.ShowID() = System.Console.Write(base.ID+”.v3”)
let laser2 = SpeedLaser()
laser2.ShowID()
let laser3 = OtherLaser()
laser3.ShowID()
```

LASER

|

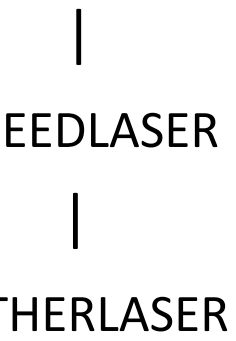
SPEEDLASER

|

OTHERLASER

What does this output?

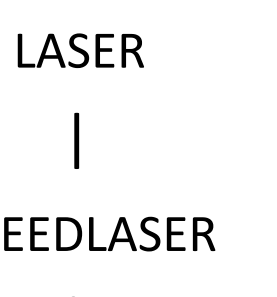
```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
type OtherLaser() =
    inherit SpeedLaser()
    override x.ShowID() = System.Console.Write(base.ID+”.v3”)
let laser2 = SpeedLaser()
laser2.ShowID()
let laser3 = OtherLaser()
laser3.ShowID()
```



Galaxy235.v2

Galaxy235.v3

```
type Laser() =
    member x.ID = "Galaxy235"
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+".v2")
type OtherLaser() =
    inherit SpeedLaser()
    override x.ShowID() = System.Console.Write(base.ID+".v3")
let laser2 = SpeedLaser()
laser2.ShowID()
let laser3 = OtherLaser()
laser3.ShowID()
```



Galaxy235.v2

Why not Galaxy235.v2.v3?

Overriding in inheritance

- A *Base* class member can be overridden **only once** in **the same** *Derived* class
- A *Base* class member can be overridden **repeatedly** in **different** *Derived* classes in **different** ways

Overriding in inheritance

- A *Base* class member can be overridden **only once** in **the same** *Derived* class
- A *Base* class member can be overridden **repeatedly** in **different** *Derived* classes in **different** ways
- When a *Base* class member is overridden in a *Derived* class, **only** the overridden version can be used in the *Derived* class

Overriding in inheritance

- A *Base* class member can be overridden **only once** in **the same** *Derived* class
- A *Base* class member can be overridden **repeatedly** in **different** *Derived* classes in **different** ways
- When a *Base* class member is overridden in a *Derived* class, **only** the overridden version can be used in the *Derived* class
- Even if a *Base* class member has been overridden, **only** the *Base* version can be used in the *Base* class

Overriding in inheritance

- A *Base* class member can be overridden **only once** in **the same** *Derived* class
- A *Base* class member can be overridden **repeatedly** in **different** *Derived* classes in **different** ways
- When a *Base* class member is overridden in a *Derived* class, **only** the overridden version can be used in the *Derived* class
- Even if a *Base* class member has been overridden, **only** the *Base* version can be used in the *Base* class
- **Cannot override constructors**

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)

let laser1 = Laser()
laser1.ShowID()                          Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                          Galaxy235.v2
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)

let laser1 = Laser()
laser1.ShowID()                                  Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                                  Galaxy235.v2
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(x.ID+”.v2”)

let laser1 = Laser()
laser1.ShowID()                          Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                          Galaxy235.v2
```

Same output

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(x.ID+”.v2”)
    member x.ID = “Cosmos000”
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(x.ID+”.v2”)
    member x.ID = “Cosmos000”
```

← **Overshadowing**

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(x.ID+”.v2”)
    member x.ID = “Cosmos000”

let laser1 = Laser()
laser1.ShowID()                          Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                          Cosmos000.v2
```

```
type Laser() =
    member x.ID = “Galaxy235”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override x.ShowID() = System.Console.Write(base.ID+”.v2”)
    member x.ID = “Cosmos000”

let laser1 = Laser()
laser1.ShowID()                          Galaxy235
let laser2 = SpeedLaser()
laser2.ShowID()                          Galaxy235.v2
```

Overshadowing (a.k.a. shadowing, hidding)

Overshadowing (a.k.a. shadowing, hidding)

- Members of a base class can be **overshadowed** in a derived class

Overshadowing (a.k.a. shadowing, hidding)

- Members of a base class can be **overshadowed** in a derived class

- Members of a base class can be **overriden** in a derived **class if they are defined "*abstract*" in the base class**

Overshadowing (a.k.a. shadowing, hidding)

- Members of a base class can be **overshadowed** in a derived class

- Members of a base class can be **overriden** in a derived **class if they are defined "*abstract*" in the base class**
- If a member in the base class is defined "*abstract*", it can be overriden but it cannot be overshadowed in the derived class

Overshadowing (a.k.a. shadowing, hidding)

- Members of a base class can be **overshadowed** in a derived class, **unless they are defined "*abstract*" in the base class**
- Members of a base class can be **overriden** in a derived **class if they are defined "*abstract*" in the base class**
- If a member in the base class is defined "*abstract*", it can be overriden but it cannot be overshadowed in the derived class

Overshadowing (a.k.a. shadowing, hidding)

- Members of a base class can be **overshadowed** in a derived class, **unless they are defined "*abstract*" in the base class**
- Members of a base class can be **overriden** in a derived **class if they are defined "*abstract*" in the base class**
- Both overriden and overshadowed members **can be inherited**

Recap today's lecture

Inheritance

- What can be inherited (and what cannot)
- Overriding inherited members
- Member definition & implementation
- Overshadowing inherited members

APPENDIX

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Accuracy = accuracy
    member private x.Power = power
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” x.Power
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
laser1.Shoot()
```

What if power is private?

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Accuracy = accuracy
    member private x.Power = power
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” x.Power
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
laser1.Shoot()
```

What if power is private?
It works

Power left: 79.000000

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Accuracy = accuracy
    member private x.Power =
        with get() = power
        and set(value) = power <- value
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” x.Power
type SpeedLaser(p, a) =
    inherit Laser(p, a)

let laser1 = SpeedLaser(80.0, 90.0)
printfn “Power left: %f” laser1.Power
```

What if power is private?
But this does not work

Error: ‘Power’ is not accessible