Programmering og Problemløsning

14 December 2018

Christina Lioma
c.lioma@di.ku.dk

Today’s lecture

Class Inheritance

- Recap last lecture
- Abstract classes
- Concrete classes
- Delegation
- Sealed classes

ADDENDUM (slide from lecture 11 Dec)

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

ADDENDUM (slide from lecture 11 Dec)

Output: float ~~or integer~~?

```
type Laser private (p, a) =
        let mutable power = p
        let mutable accuracy = a
        member x.Shoot() =
                power <- power – 1.0                ← FLOAT
                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 ← **UNIT**

```
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()
    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”)
    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(base.ID+”.v2”)
    member x.ID = “Cosmos000”          ← Overshadowing
```

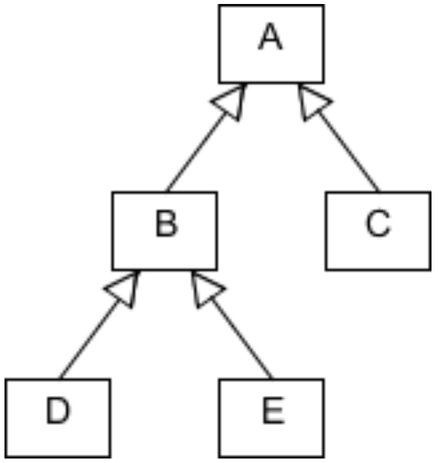
Q: Can I overshadow a method?

A: yes (unless it is declared abstract in base class)

Q: Are overshadowed members inherited?

A: yes (but be aware of the scope)

Inheritance creates class hierarchies



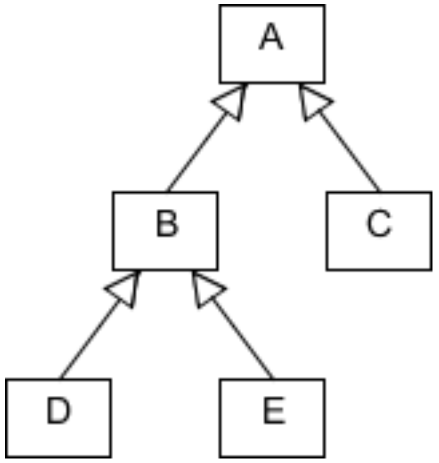
Every .NET class (incl. primitive data types) participates in inheritance

Classes close to the top tend to be general

Classes close to the bottom tend to be specialised

The further up, the more general the classes

Inheritance creates class hierarchies



Every .NET class (incl. primitive data types) participates in inheritance

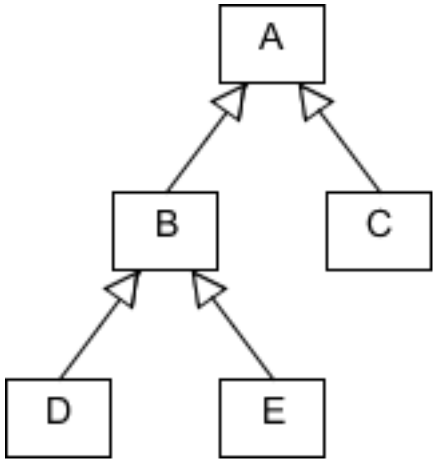
Classes close to the top tend to be general

Classes close to the bottom tend to be specialised

The further up, the more general the classes

Abstract classes (typically top of hierarchy)

Inheritance creates class hierarchies



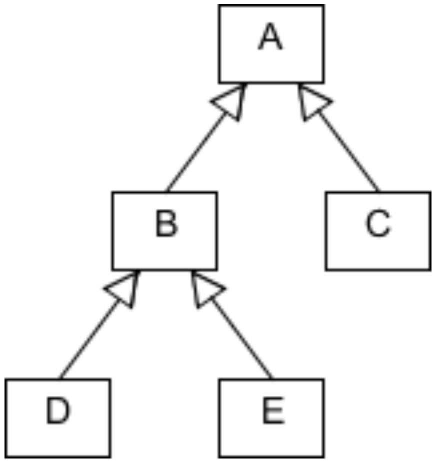
Every .NET class (incl. primitive data types) participates in inheritance

Classes close to the top tend to be general
Classes close to the bottom tend to be specialised
The further up, the more general the classes

Abstract classes (typically top of hierarchy):

- *Cannot be instantiated*

Inheritance creates class hierarchies



Every .NET class (incl. primitive data types) participates in inheritance

Classes close to the top tend to be general
Classes close to the bottom tend to be specialised
The further up, the more general the classes

Abstract classes (typically top of hierarchy):

- *Cannot be instantiated directly*
- *Accessible only through derived classes*
- *Contain members without an implementation*

```
[<AbstractClass>]
type Laser() =
      abstract member ID : string
      abstract member ShowID : unit -> unit
```

```
[<AbstractClass>]
type Laser() =
    abstract member ID : string
    abstract member ShowID : unit -> unit
```

→ ABSTRACT CLASS

```
[<AbstractClass>]                            → ABSTRACT CLASS
type Laser() =
    abstract member ID : string              → DEFINITION
    abstract member ShowID : unit -> unit    → DEFINITION
```

```
[<AbstractClass>]                                  → ABSTRACT CLASS
type Laser() =
     abstract member ID : string                   → DEFINITION
     abstract member ShowID : unit -> unit         → DEFINITION
type SpeedLaser() =
     inherit Laser()
     override x.ID = “Galaxy”
     override x.ShowID() = System.Console.Write(x.ID)
```

```
[<AbstractClass>]                                → ABSTRACT CLASS
type Laser() =
      abstract member ID : string                → DEFINITION
      abstract member ShowID : unit -> unit      → DEFINITION
type SpeedLaser() =
      inherit Laser()
      override x.ID = "Galaxy"                             → IMPL
      override x.ShowID() = System.Console.Write(x.ID)     → IMPL
```

```
[<AbstractClass>]                                  → ABSTRACT CLASS
type Laser() =
      abstract member ID : string                  → DEFINITION
      abstract member ShowID : unit -> unit        → DEFINITION
type SpeedLaser() =
      inherit Laser()
      override x.ID = “Galaxy”                                  → IMPL
      override x.ShowID() = System.Console.Write(x.ID)          → IMPL



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

```
[<AbstractClass>]                                        → ABSTRACT CLASS
type Laser() =
    abstract member ID : string                          → DEFINITION
    abstract member ShowID : unit -> unit                → DEFINITION
type SpeedLaser() =
    inherit Laser()
    override x.ID = “Galaxy”                                     → IMPL
    override x.ShowID() = System.Console.Write(x.ID)             → IMPL

let laser1 = new Laser()
laser1.ShowID()                                  output?
let laser2 = new SpeedLaser()
laser2.ShowID()                                  Galaxy
```

```
[<AbstractClass>]                              → ABSTRACT CLASS
type Laser() =
      abstract member ID : string              → DEFINITION
      abstract member ShowID : unit -> unit    → DEFINITION
type SpeedLaser() =
      inherit Laser()
      override x.ID = "Galaxy"                                   → IMPL
      override x.ShowID() = System.Console.Write(x.ID)           → IMPL

let laser1 = new Laser()
laser1.ShowID()                                Does not run
let laser2 = new SpeedLaser()
laser2.ShowID()                                Galaxy
```

"Instances of this type cannot be created since it has been marked abstract"

```
[<AbstractClass>]                                → ABSTRACT CLASS
type Laser() =
    abstract member ID : string                  → DEFINITION
    abstract member ShowID : unit -> unit        → DEFINITION
type SpeedLaser() =
    inherit Laser()
    override x.ID = “Galaxy”                               → IMPL
    override x.ShowID() = System.Console.Write(x.ID)       → IMPL
```

Abstract class:

- **Cannot be instantiated**
- **Accessed only from Derived**
- **Contains unimplemented members**

```
[<AbstractClass>]                                → ABSTRACT CLASS
type Laser() =
      abstract member ID : string                → DEFINITION
      abstract member ShowID : unit -> unit      → DEFINITION
type SpeedLaser() =
      inherit Laser()
      override x.ID = “Galaxy”                              → IMPL
      override x.ShowID() = System.Console.Write(x.ID)      → IMPL
                                                ↑
```

Abstract class: ***Why not base.ID?***

- **Cannot be instantiated**
- **Accessed only from Derived**
- **Contains unimplemented members**

```
[<AbstractClass>]                                  → ABSTRACT CLASS
type Laser() =
    abstract member ID : string                    → DEFINITION
    abstract member ShowID : unit -> unit          → DEFINITION
type SpeedLaser() =
    inherit Laser()
    override x.ID = “Galaxy”                                   → IMPL
    override x.ShowID() = System.Console.Write(x.ID)           → IMPL
type OtherLaser() =
    inherit Laser()
    default x.ID = “Galaxy”                                    → IMPL
    default x.ShowID() = System.Console.Write(x.ID)            → IMPL
```

```
[<AbstractClass>]                                  → ABSTRACT CLASS
type Laser() =
    abstract member ID : string                    → DEFINITION
    abstract member ShowID : unit -> unit          → DEFINITION
type SpeedLaser() =
    inherit Laser()
    override x.ID = “Galaxy”                                 → IMPL
    override x.ShowID() = System.Console.Write(x.ID)         → IMPL
type OtherLaser() =
    inherit Laser()
    default x.ID = “Galaxy”                                  → IMPL
    default x.ShowID() = System.Console.Write(x.ID)          → IMPL
```

```
[<AbstractClass>]                              → ABSTRACT CLASS
type Laser() =
    abstract member ID : string                → DEFINITION
    abstract member ShowID : unit -> unit      → DEFINITION
type SpeedLaser() =
    inherit Laser()
    override x.ID = “Galaxy”                              → IMPL
    override x.ShowID() = System.Console.Write(x.ID)      → IMPL
type OtherLaser() =
    inherit Laser()
    default x.ID = “Galaxy”                               → IMPL
    default x.ShowID() = System.Console.Write(x.ID)       → IMPL
```

BOTH ARE VALID: When inheriting from *abstract* base class, *override* and *default* can be used interchangeably

```
[<AbstractClass>]
type Laser() =
      abstract member ID : string
      abstract member ShowID : unit -> unit
type SpeedLaser() =
      inherit Laser()
      override x.ID = “Galaxy”
      override x.ShowID() = System.Console.Write(x.ID)
type OtherLaser() =
      inherit Laser()
      default x.ID = “Galaxy”
      default x.ShowID() = System.Console.Write(x.ID)
```

Convention:

- Use *override* in derived class
- Use *default* in base class

BOTH ARE VALID: When inheriting from *abstract* base class, *override* and *default* can be used interchangeably

AN ABSTRACT CLASS:

```
[<AbstractClass>]
type LaserA() =
    abstract member ID : string
    abstract member ShowID : unit -> unit
```

AN ABSTRACT CLASS:

```
[<AbstractClass>]
type LaserA() =
      abstract member ID : string                         -> DEF
      abstract member ShowID : unit -> unit               -> DEF
```

AN ABSTRACT CLASS:

```
[<AbstractClass>]
type LaserA() =
    abstract member ID : string                          -> DEF
    abstract member ShowID : unit -> unit                -> DEF

type LaserB() =
    member x.ID = “Galaxy”                               -> DEF & IMPL
    member x.ShowID() = System.Console.Write(x.ID)       -> DEF & IMPL
```

AN ABSTRACT CLASS:

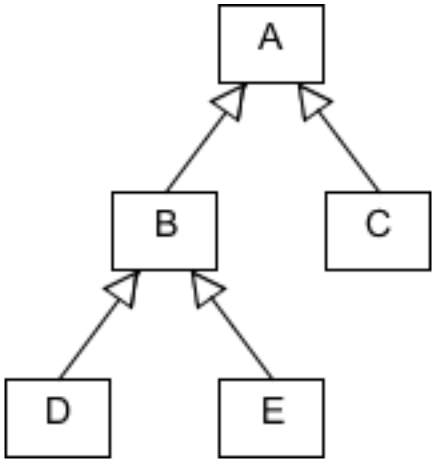
```
[<AbstractClass>]
type LaserA() =
    abstract member ID : string                          -> DEF
    abstract member ShowID : unit -> unit                -> DEF
```

A CONCRETE CLASS:

```
type LaserB() =
    member x.ID = “Galaxy”                               -> DEF & IMPL
    member x.ShowID() = System.Console.Write(x.ID)       -> DEF & IMPL
```

AN ABSTRACT CLASS:

```
[<AbstractClass>]
type LaserA() =
    abstract member ID : string                         -> DEF
    abstract member ShowID : unit -> unit               -> DEF
```



A CONCRETE CLASS:

```
type LaserB() =
    member x.ID = “Galaxy”                              -> DEF & IMPL
    member x.ShowID() = System.Console.Write(x.ID)      -> DEF & IMPL
```

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*

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*

Three steps to override an inherited member:

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

  use keyword *abstract*

- *If the base class is concrete*, 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*

abstract: potential source of confusion

abstract: potential source of confusion

We have seen so far:

- *<u>Abstract data types:</u> we invent them*

abstract: potential source of confusion

We have seen so far:

- *<u>Abstract data types:</u> we invent them*
- *<u>Abstract class member:</u> can be overridden*

abstract: potential source of confusion

We have seen so far:

- *Abstract data types:* *we invent them*
- *Abstract class member:* *can be overridden*
- *Abstract class:* *contains members that do not have an implementation*

abstract: potential source of confusion

We have seen so far:

- *Abstract data types:* *we invent them*
- *Abstract class member:* *can be overridden*
- *Abstract class:* *contains members that do not have an implementation*

This does not mean that only abstract classes have abstract members

abstract: potential source of confusion

We have seen so far:

- *Abstract data types:* *we invent them*
- *Abstract class member:* *can be overridden*
- *Abstract class:* *contains members that do not have an implementation*

This does not mean that only abstract classes have abstract members

Concrete classes can also have abstract members

AN ABSTRACT CLASS

```
[<AbstractClass>]
type Laser1() =
      abstract member ID : string                -> DEF
      abstract member ShowID : unit -> unit  -> DEF
```

A CONCRETE CLASS

```
type Laser2() =
      member x.ID = "Galaxy"                                      -> DEF & IMPL
      member x.ShowID() = System.Console.Write(x.ID)      -> DEF & IMPL
```

AN ABSTRACT CLASS

```
[<AbstractClass>]
type Laser1() =
      abstract member ID : string               -> DEF
      abstract member ShowID : unit -> unit  -> DEF
```

CONCRETE CLASSES

```
type Laser2() =
      member x.ID = “Galaxy”                              -> DEF & IMPL
      member x.ShowID() = System.Console.Write(x.ID)      -> DEF & IMPL
type Laser3() =
      abstract member ID : string                         -> DEF
      default x.ID = “Galaxy”                             -> IMPL
      abstract member ShowID : unit -> unit               -> DEF
      default x.ShowID() = System.Console.Write(x.ID)     -> IMPL
```

AN ABSTRACT CLASS

```
[<AbstractClass>]
type Laser1() =
        abstract member ID : string                 -> DEF
        abstract member ShowID : unit -> unit   -> DEF
```

CONCRETE CLASSES

```
type Laser2() =
        member x.ID = “Galaxy”                                   -> DEF & IMPL
        member x.ShowID() = System.Console.Write(x.ID)           -> DEF & IMPL
type Laser3() =
        abstract member ID : string                              -> DEF
        default x.ID = “Galaxy”                                  -> IMPL
        abstract member ShowID : unit -> unit                    -> DEF
        default x.ShowID() = System.Console.Write(x.ID)          -> IMPL
type Laser4() =
        member x.ID = “Galaxy”                                   -> DEF & IMPL
        abstract member ShowID : unit -> unit                    -> DEF
        default x.ShowID() = System.Console.Write(x.ID)          -> IMPL
```

AN ABSTRACT CLASS

```
[<AbstractClass>]
type Laser1() =
        abstract member ID : string               -> DEF
        abstract member ShowID : unit -> unit  -> DEF
        member x.SayHi() = printfn “Hi”            -> DEF & IMPL
```

CONCRETE CLASSES

```
type Laser2() =
        member x.ID = “Galaxy”                                    -> DEF & IMPL
        member x.ShowID() = System.Console.Write(x.ID)    -> DEF & IMPL
type Laser3() =
        abstract member ID : string                               -> DEF
        default x.ID = “Galaxy”                                   -> IMPL
        abstract member ShowID : unit -> unit                     -> DEF
        default x.ShowID() = System.Console.Write(x.ID)     -> IMPL
type Laser4() =
        member x.ID = “Galaxy”                                    -> DEF & IMPL
        abstract member ShowID : unit -> unit                     -> DEF
        default x.ShowID() = System.Console.Write(x.ID)     -> IMPL
```

AN ABSTRACT CLASS HAS <u>AT LEAST 1 MEMBER WITHOUT IMPLEMENTATION</u>

```
[<AbstractClass>]
type Laser1() =
        abstract member ID : string                 -> DEF
        abstract member ShowID : unit -> unit       -> DEF
        member x.SayHi() = printfn “Hi”             -> DEF & IMPL
```

CONCRETE CLASSES

```
type Laser2() =
        member x.ID = “Galaxy”                              -> DEF & IMPL
        member x.ShowID() = System.Console.Write(x.ID)      -> DEF & IMPL
type Laser3() =
        abstract member ID : string                         -> DEF
        default x.ID = “Galaxy”                             -> IMPL
        abstract member ShowID : unit -> unit               -> DEF
        default x.ShowID() = System.Console.Write(x.ID)     -> IMPL
type Laser4() =
        member x.ID = “Galaxy”                              -> DEF & IMPL
        abstract member ShowID : unit -> unit               -> DEF
        default x.ShowID() = System.Console.Write(x.ID)     -> IMPL
```

AN ABSTRACT CLASS HAS AT LEAST 1 MEMBER WITHOUT IMPLEMENTATION

```
[<AbstractClass>]
type Laser1() =
        abstract member ID : string                 -> DEF
        abstract member ShowID : unit -> unit  -> DEF
        member x.SayHi() = printfn “Hi”           -> DEF & IMPL
```

A CONCRETE CLASS HAS DEFINITIONS & IMPLEMENTATIONS FOR ALL MEMBERS

```
type Laser2() =
        member x.ID = “Galaxy”                              -> DEF & IMPL
        member x.ShowID() = System.Console.Write(x.ID)      -> DEF & IMPL
type Laser3() =
        abstract member ID : string                         -> DEF
        default x.ID = “Galaxy”                             -> IMPL
        abstract member ShowID : unit -> unit               -> DEF
        default x.ShowID() = System.Console.Write(x.ID)     -> IMPL
type Laser4() =
        member x.ID = “Galaxy”                              -> DEF & IMPL
        abstract member ShowID : unit -> unit               -> DEF
        default x.ShowID() = System.Console.Write(x.ID)     -> IMPL
```

AN ABSTRACT CLASS HAS AT LEAST 1 MEMBER WITHOUT IMPLEMENTATION

```
[<AbstractClass>]
type Laser1() =
    abstract member ID : string             -> DEF
    abstract member ShowID : unit -> unit   -> DEF
    member x.SayHi() = printfn "Hi"         -> DEF & IMPL
```

A CONCRETE CLASS HAS DEFINITIONS & IMPLEMENTATIONS FOR ALL MEMBERS

MEMBERS CAN BE ABSTRACT

```
type Laser2() =
    member x.ID = "Galaxy"                               -> DEF & IMPL
    member x.ShowID() = System.Console.Write(x.ID)       -> DEF & IMPL
type Laser3() =
    abstract member ID : string                          -> DEF
    default x.ID = "Galaxy"                              -> IMPL
    abstract member ShowID : unit -> unit                -> DEF
    default x.ShowID() = System.Console.Write(x.ID)      -> IMPL
type Laser4() =
    member x.ID = "Galaxy"                               -> DEF & IMPL
    abstract member ShowID : unit -> unit                -> DEF
    default x.ShowID() = System.Console.Write(x.ID)      -> IMPL
```

The type of class (abstract or concrete) does not affect overriding

The type of class (abstract or concrete) does not affect overriding

We can override:

- A *Base* class member of an abstract class that has no implementation

The type of class (abstract or concrete) does not affect overriding

We can override:

- A *Base* class member of an abstract class that has no implementation
- A *Base* class member of a concrete class that has an implementation

The type of class (abstract or concrete) does not affect overriding

We can override:

- A *Base* class member of an abstract class that has no implementation
- A *Base* class member of a concrete class that has an implementation

We can override *any* class member, *as long as it is marked* ***abstract member***

Abstract Classes

- Typically higher in class hierarchy
- Contain at least 1 member without an implementation
- Cannot be instantiated directly
- Accessible only through derived classes

Abstract Classes

- Typically higher in class hierarchy
- Contain at least 1 member without an implementation
- Cannot be instantiated directly
- Accessible only through derived classes ***or through delegation***

```
[<AbstractClass>]
type Laser() =
    abstract member ID : string
    member x.ShowID() = System.Console.Write(x.ID)
```

```
[<AbstractClass>]
type Laser() =
      abstract member ID : string
      member x.ShowID() = System.Console.Write(x.ID)

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

ERROR: Instances of this type cannot be created since it has been marked abstract or not all methods have been given implementations.

```
[<AbstractClass>]
type Laser() =
    abstract member ID : string
    member x.ShowID() = System.Console.Write(x.ID)

let laser1 = { Laser() with
    member x.ID = “Galaxy” }
laser1.ShowID()
```

```
[<AbstractClass>]
type Laser() =
    abstract member ID : string
    member x.ShowID() = System.Console.Write(x.ID)

let laser1 = { Laser() with
    member x.ID = "Galaxy" }
laser1.ShowID()
```

DELEGATION

```
[<AbstractClass>]
type Laser() =
    abstract member ID : string
    member x.ShowID() = System.Console.Write(x.ID)

let laser1 = { Laser() with                    DELEGATION
    member x.ID = “Galaxy” }
laser1.ShowID()
```

output: “Galaxy”

```
[<AbstractClass>]
type Laser() =
      abstract member ID : string
      member x.ShowID() = System.Console.Write(x.ID)

let laser1 = { Laser() with                    DELEGATION
      member x.ID = "Galaxy" }
laser1.ShowID()
```

output: "Galaxy"

We can instantiate a partially implemented abstract class with delegation

```
[<AbstractClass>]
type Laser() =
    abstract member ID : string
    abstract member ShowID : unit -> unit
```

```
[<AbstractClass>]
type Laser() =
    abstract member ID : string
    abstract member ShowID : unit -> unit
```

Can we instantiate a fully unimplemented abstract class with delegation?

```
[<AbstractClass>]
type Laser() =
      abstract member ID : string
      abstract member ShowID : unit -> unit

let laser1 = { Laser() with
      member x.ID = “Galaxy”
      member x.ShowID() = System.Console.Write(x.ID) }
laser1.ShowID()
```

```
[<AbstractClass>]
type Laser() =
      abstract member ID : string
      abstract member ShowID : unit -> unit

let laser1 = { Laser() with
      member x.ID = “Galaxy”
      member x.ShowID() = System.Console.Write(x.ID) }
laser1.ShowID()
```

output: “Galaxy”

```
[<AbstractClass>]
type Laser() =
    abstract member ID : string
    abstract member ShowID : unit -> unit

let laser1 = { Laser() with
    member x.ID = “Galaxy”
    member x.ShowID() = System.Console.Write(x.ID) }
laser1.ShowID()
```

output: “Galaxy”

We can instantiate a fully unimplemented abstract class with delegation

```
~~[<AbstractClass>]~~
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
```

```
~~[<AbstractClass>]~~
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
```

Can we instantiate a concrete class with delegation?

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

let laser1 = { new Laser() with
    member x.ShowID() = System.Console.Write(x.ID+”.v2”)}
laser1.ShowID()
```

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

let laser1 = { new Laser() with
      member x.ShowID() = System.Console.Write(x.ID+”.v2”)}
laser1.ShowID()
```

output: “Galaxy.v2”

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

let laser1 = { new Laser() with
    member x.ShowID() = System.Console.Write(x.ID+”.v2”)}
laser1.ShowID()
```

output: “Galaxy.v2”

We can instantiate a concrete class with delegation

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

let laser1 = { new Laser() with
    member x.ID = “Orbit” }
laser1.ShowID()
```

output?

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

let laser1 = { new Laser() with
    member x.ID = “Orbit” }
laser1.ShowID()
```

Error: ID not available to implement

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

let laser1 = { new Laser() with
      member x.ShowID() = System.Console.Write(x.ID+”.v2”)}
laser1.ShowID()
```

output: “Galaxy”

We can instantiate a concrete class with delegation only for members that can be overridden. Cannot overshadow with delegation.

Delegation

- Specify implementation during instantiation

Delegation

- Specify implementation during instantiation

Can be done when:

- there is no implementation

Delegation

- Specify implementation during instantiation

Can be done when:

- there is no implementation
- there is an implementation that can be overridden

Delegation

- Specify implementation during instantiation

Can be done when:

- there is no implementation
- there is an implementation that can be overridden

```
let instanceName =
{ new ClassName() with implementation }
```

Delegation

- Specify implementation during instantiation

Can be done when:

- there is no implementation
- there is an implementation that can be overridden

```
let instanceName =
{ new ClassName() with implementation }
```

keyword "new" seems to be:

- **optional** when delegating from **abstract** class
- **compulsory** when delegating from **concrete** class

Abstract classes: **must*** be inherited by other classes

Concrete classes: **can** be inherited by other classes

*exception: delegation

Abstract classes: **must*** be inherited by other classes

Concrete classes: **can** be inherited by other classes

Sealed classes: **cannot** be inherited by other classes

*exception: delegation

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

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

```
[<Sealed>]
type Laser() =
    member x.ID = “Galaxy”
    member x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()

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

Error: Cannot inherit a sealed type

```
[<Sealed>]
type Laser() =
    member x.ID = “Galaxy”
    member x.ShowID() = System.Console.Write(x.ID)
~~type SpeedLaser() =~~
    ~~inherit Laser()~~

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

```
[<Sealed>]
type Laser() =
    member x.ID = “Galaxy”
    member x.ShowID() = System.Console.Write(x.ID)
~~type SpeedLaser() =~~
    ~~inherit Laser()~~

let laser1 = new Laser()
laser1.ShowID()                                    Galaxy
~~let laser2 = new SpeedLaser()~~
~~laser2.ShowID()~~
```

```
[<Sealed>]
type Laser() =
    member x.ID = “Galaxy”
    member x.ShowID() = System.Console.Write(x.ID)
~~type SpeedLaser() =~~
    ~~inherit Laser()~~

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

This is a concrete class that is also sealed

Galaxy

Abstract classes: **must** be inherited

Concrete classes: **can** be inherited

Sealed classes: **cannot** be inherited

Abstract classes: **must** be inherited

Concrete classes:
- Some **can** be inherited
- Some **cannot** be inherited → Sealed classes

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)

```
[<Sealed>]
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
```

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)

```
[<Sealed>]
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
let laser1 = { new Laser() with
    member x.ShowID() = System.Console.Write(x.ID+”.v2”) }
laser1.ShowID()
```

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)

```
[<Sealed>]
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit          CAN OVERRIDE
    default x.ShowID() = System.Console.Write(x.ID)
let laser1 = { new Laser() with
    member x.ShowID() = System.Console.Write(x.ID+”.v2”) }
laser1.ShowID()
```

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)

```
[<Sealed>]
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit              CAN OVERRIDE
    default x.ShowID() = System.Console.Write(x.ID)
let laser1 = { new Laser() with                        OVERRIDE
    member x.ShowID() = System.Console.Write(x.ID+”.v2”) }
laser1.ShowID()
```

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)

```
[<Sealed>]
type Laser() =
     member x.ID = “Galaxy”
     abstract member ShowID : unit -> unit
     default x.ShowID() = System.Console.Write(x.ID)
let laser1 = { new Laser() with
     member x.ShowID() = System.Console.Write(x.ID+”.v2”) }
laser1.ShowID()
```

Error:
“Cannot create an extension of a sealed type”

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)
- **Cannot be instantiated with delegation** (cannot override the implementation of their members)

Error:
"Cannot create an extension of a sealed type"

```
[<Sealed>]
type Laser() =
    member x.ID = "Galaxy"
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
let laser1 = { new Laser() with
    member x.ShowID() = System.Console.Write(x.ID+".v2") }
laser1.ShowID()
```

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)
- **Cannot be instantiated with delegation** (cannot override the implementation of their members)

What about this?

```
[<Sealed>]
type Laser() =
    member x.ID = "Galaxy"
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
let laser1 =  new Laser()

laser1.ShowID()
```

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)
- **Cannot be instantiated with delegation** (cannot override the implementation of their members)

Will run, but making ShowID abstract is pointless

```
[<Sealed>]
type Laser() =
    member x.ID = "Galaxy"
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
let laser1 =  new Laser()

laser1.ShowID()
```

Sealed classes:

- **Cannot be inherited** (other classes cannot derive from sealed)
- Can only be concrete (**all** their members must have implementations)
- **Cannot be instantiated with delegation** (cannot override the implementation of their members)

```
[<Sealed>]
type Laser() =
    member x.ID = "Galaxy"
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
let laser1 =  new Laser()

laser1.ShowID()
```

Will run, but making ShowID abstract is pointless

LOGICAL ERROR

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

```
type Laser() =
        member x.ID = “Galaxy”
        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.Accuracy = 80
```

```
type Laser() =
        member x.ID = “Galaxy”
        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.Accuracy = 80
type DistanceLaser() =
        inherit SpeedLaser()
        abstract member Range : int
```

```
type Laser() =
        member x.ID = “Galaxy”
        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.Accuracy = 80
type DistanceLaser() =
        inherit SpeedLaser()
        abstract member Range : int
[<Sealed>]
type GigaLaser() =
        inherit SpeedLaser()
```

```
type Laser() =
    member x.ID = “Galaxy”
    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.Accuracy = 80
type DistanceLaser() =
    inherit SpeedLaser()
    abstract member Range : int
[<Sealed>]
type GigaLaser() =
    inherit SpeedLaser()
```

$$\begin{tikzcd}
 & \text{Laser} \arrow[d, dash] & \\
 & \text{SpeedLaser} \arrow[dl, dash] \arrow[dr, dash] & \\
\text{DistanceLaser} & & \text{GigaLaser}
\end{tikzcd}$$

```
type Laser() =
        member x.ID = “Galaxy”
        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.Accuracy = 80
type DistanceLaser() =
        inherit SpeedLaser()
        abstract member Range : int
[<Sealed>]
type GigaLaser() =
        inherit SpeedLaser()
let laser1 = GigaLaser()
laser1.ShowID()
```

```
          Laser
            |
        SpeedLaser
        |         |
DistanceLaser  GigaLaser
```

```
type Laser() =
        member x.ID = “Galaxy”
        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.Accuracy = 80
type DistanceLaser() =
        inherit SpeedLaser()
        abstract member Range : int
[<Sealed>]
type GigaLaser() =
        inherit SpeedLaser()
let laser1 = GigaLaser()
laser1.ShowID()
```

```
             Laser
               |
           SpeedLaser
        |              |
DistanceLaser  GigaLaser
```

What does this output?

```
type Laser() =
        member x.ID = “Galaxy”
        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.Accuracy = 80
type DistanceLaser() =
        inherit SpeedLaser()
        abstract member Range : int
[<Sealed>]
type GigaLaser() =
        inherit SpeedLaser()
let laser1 = GigaLaser()
laser1.ShowID()
```

$$\begin{tikzcd} & \text{Laser} \arrow[d, dash] & \\ & \text{SpeedLaser} \arrow[dl, dash] \arrow[dr, dash] & \\ \textbf{DistanceLaser} & & \text{GigaLaser} \end{tikzcd}$$

ERROR: “No implementation was given for 'abstract member DistanceLaser.Range : int'”

```
type Laser() =
        member x.ID = “Galaxy”
        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.Accuracy = 80
type DistanceLaser() =
        inherit SpeedLaser()
        abstract member Range : int
[<Sealed>]
type GigaLaser() =
        inherit SpeedLaser()
let laser1 = GigaLaser()
let laser2 = { new DistanceLaser() with member x.Range = 10 }
laser1.ShowID()
```

$$\begin{tikzcd}
& \text{Laser} \arrow[d, no head] & \\
& \text{SpeedLaser} \arrow[dl, no head] \arrow[dr, no head] & \\
\text{DistanceLaser} & & \text{GigaLaser}
\end{tikzcd}$$

```
type Laser() =
      member x.ID = “Galaxy”
      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.Accuracy = 80
type DistanceLaser() =
      inherit SpeedLaser()
      abstract member Range : int
[<Sealed>]
type GigaLaser() =
      inherit SpeedLaser()
let laser1 = GigaLaser()
let laser2 = { new DistanceLaser() with member x.Range = 10 }
laser1.ShowID()
```

$$\begin{tikzcd}
 & \text{Laser} \arrow[d, dash] & \\
 & \text{SpeedLaser} \arrow[dl, dash] \arrow[dr, dash] & \\
\text{DistanceLaser} & & \text{GigaLaser}
\end{tikzcd}$$

ERROR: “No implementation was given for 'abstract member DistanceLaser.Range : int'”

```
type Laser() =
        member x.ID = "Galaxy"
        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.Accuracy = 80
type DistanceLaser() =
        inherit SpeedLaser()
        abstract member Range : int
[<Sealed>]
type GigaLaser() =
        inherit SpeedLaser()
let laser1 = GigaLaser()
let laser2 = { new DistanceLaser() with member x.Range = 10 }
laser1.ShowID()
```

$$\begin{tikzcd}
 & \text{Laser} \arrow[d, dash] & \\
 & \text{SpeedLaser} \arrow[dl, dash] \arrow[dr, dash] & \\
\text{DistanceLaser} & & \text{GigaLaser}
\end{tikzcd}$$

ERROR SHOULD BE: "you have an abstract class that you have not declared abstract'"

```
type Laser() =
      member x.ID = “Galaxy”
      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.Accuracy = 80
[<AbstractClass>]
type DistanceLaser() =
      inherit SpeedLaser()
      abstract member Range : int
[<Sealed>]
type GigaLaser() =
      inherit SpeedLaser()
let laser1 = GigaLaser()
let laser2 = { new DistanceLaser() with member x.Range = 10 }
laser1.ShowID()
```

```
           Laser
             |
        SpeedLaser
         |        |
DistanceLaser  GigaLaser
```

Galaxy.v2

```
type Laser() =
        member x.ID = “Galaxy”
        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.Accuracy = 80
[<AbstractClass>]
type DistanceLaser() =
        inherit SpeedLaser()
        abstract member Range : int
[<Sealed>]
type GigaLaser() =
        inherit SpeedLaser()
let laser1 = GigaLaser()

laser1.ShowID()
```

```
                 Laser
               (concrete)
                   |
               SpeedLaser
               (concrete)
               |         |
     DistanceLaser  GigaLaser
     (abstract)       (concrete, sealed)
```

Galaxy.v2

Recap today’s lecture

Class inheritance

- Abstract classes
- Concrete classes
- Delegation (implementation during instantiation)
- Sealed classes