Programmering og Problemløsning

20 December 2016

Christina Lioma

c.lioma@di.ku.dk

Today's lecture

- Class inheritance
  - Overriding
  - Abstract & concrete classes
  - Delegation
  - Sealed classes

Create a Class Animal(weight, maxSpeed) or Animal(maxSpeed)

Create a Class Animal(weight, maxSpeed) or Animal(maxSpeed)
Attributes: **Weight, MaxSpeed**, FoodNeeded, CurrentSpeed

Create a Class Animal(weight, maxSpeed) or Animal(maxSpeed)
Attributes: Weight, MaxSpeed, FoodNeeded, CurrentSpeed

Methods:

- GetCurrentSpeed from (a) FoodEaten with respect to FoodNeeded and (b) MaxSpeed

Create a Class Animal(weight, maxSpeed) or Animal(maxSpeed)
Attributes: Weight, MaxSpeed, FoodNeeded, CurrentSpeed

Methods:

- GetCurrentSpeed from (a) FoodEaten with respect to FoodNeeded and (b) MaxSpeed

- GetFoodNeeded with respect to Weight

Create a Class Animal(weight, maxSpeed) or Animal(maxSpeed)
Attributes: Weight, MaxSpeed, FoodNeeded, CurrentSpeed

Methods:

- GetCurrentSpeed from (a) FoodEaten with respect to FoodNeeded and (b) MaxSpeed

- GetFoodNeeded with respect to Weight

[Above we have no value for FoodEaten]

Create a Class Animal(weight, maxSpeed) or Animal(maxSpeed)
Attributes: Weight, MaxSpeed, FoodNeeded, CurrentSpeed

Methods:

- GetCurrentSpeed from (a) FoodEaten with respect to FoodNeeded and (b) MaxSpeed

- GetFoodNeeded with respect to Weight

[Above we have no value for FoodEaten]

Create Class Carnivore that inherits Animal & overrides GetFoodNeeded

Create Class Herbivore that inherits Animal & overrides GetFoodNeeded

Create a Class Animal(weight, maxSpeed) or Animal(maxSpeed)
Attributes: Weight, MaxSpeed, FoodNeeded, CurrentSpeed

Methods:

- GetCurrentSpeed from (a) FoodEaten with respect to FoodNeeded and (b) MaxSpeed

- GetFoodNeeded with respect to Weight

[Above we have no value for FoodEaten]

Create Class Carnivore that inherits Animal & overrides GetFoodNeeded

Create Class Herbivore that inherits Animal & overrides GetFoodNeeded

Cheetah is instance of Carnivore

Antelope and Wildebeest are instances of Herbivore

Create a Class Animal(weight, maxSpeed) or Animal(maxSpeed)
Attributes: Weight, MaxSpeed, FoodNeeded, CurrentSpeed

Methods:

- GetCurrentSpeed from (a) FoodEaten with respect to FoodNeeded and (b) MaxSpeed

- GetFoodNeeded with respect to Weight

[Above we have no value for FoodEaten]

Create Class Carnivore that inherits Animal & overrides GetFoodNeeded

Create Class Herbivore that inherits Animal & overrides GetFoodNeeded

Cheetah is instance of Carnivore

Antelope and Wildebeest are instances of Herbivore

***Each instance* generates a random percentage of FoodEaten with respect to FoodNeeded. This is how we get the value for FoodEaten.**

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

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

  use keyword *override*

```
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”)
```

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”)
```

DEFINITION

IMPLEMENTATION

NEW IMPLEMENTATION

“**base**” keyword:

- accesses directly members of the base class
- only available in classes that **explicitly inherit** from another class

```
[<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)

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

```
[<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)

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

can we use ***base.ID*** instead?

↓

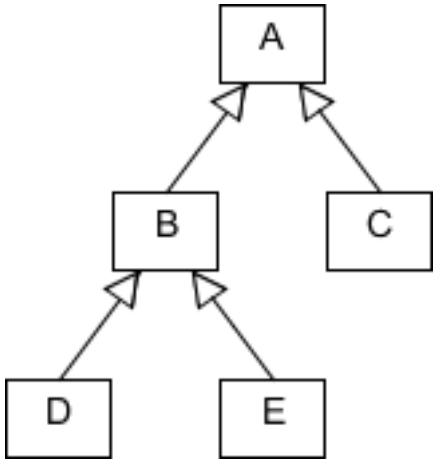
Galaxy

```
[<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)

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

can we use ***base.ID*** instead?

Error:” Cannot call an abstract base member: ID”

Inheritance and Abstract Classes

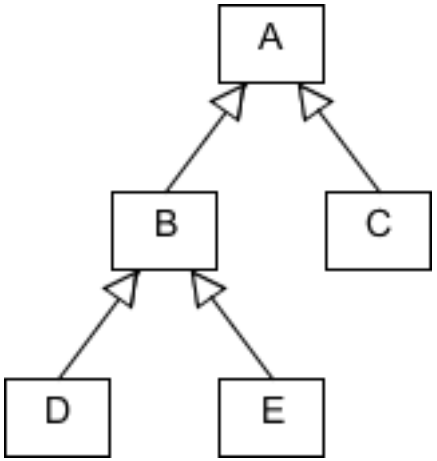


Abstract classes (typically higher in class hierarchy):

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

AN ABSTRACT CLASS:

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



A CONCRETE CLASS:

```
type Laser() =
    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 ***if the base is concrete***

  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:

- *Abstract data types: we invent them*

abstract: potential source of confusion

We have seen so far:

- *Abstract data types: we invent them*
- *Abstract class member: 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 at least 1 member that does 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 at least 1 member that does 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 at least 1 member that does 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 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
```

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 (OVERRIDABLE)

```
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 **if it is marked abstract**

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 **if it is marked abstract**
- A *Base* class member of a **concrete** class that has an implementation **if it is marked abstract**

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()
```

```
[<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
    member x.ID = "Galaxy" }
laser1.ShowID()
```

DELEGATION

output: "Galaxy"

```
[<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

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 override or 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

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

Abstract classes: must be inherited by other classes

Concrete classes: can be inherited by other classes

Sealed classes: cannot be inherited by other classes

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

```
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()
```

```
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()
```

```
[<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

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”
    member 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”
    member x.ShowID() = System.Console.Write(x.ID)
let laser1 = new Laser()
laser1.ShowID()                    Galaxy
```

Sealed classes:

- Cannot be inherited
- 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
- 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
- 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()
```

CAN OVERRIDE

Sealed classes:

- Cannot be inherited
- 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()
```

CAN OVERRIDE

OVERRIDE

Sealed classes:

- Cannot be inherited
- 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
- 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
- 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
- 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
- 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

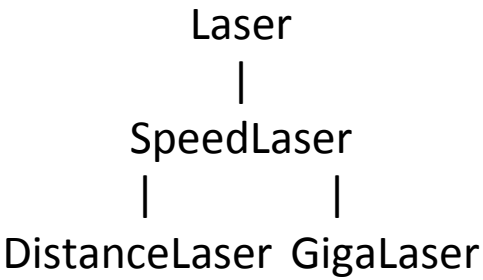
```
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
```

```
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override ShowID() = System.Console.Write(base.ID+”.v2”)
    member x.Accuracy = 80
```

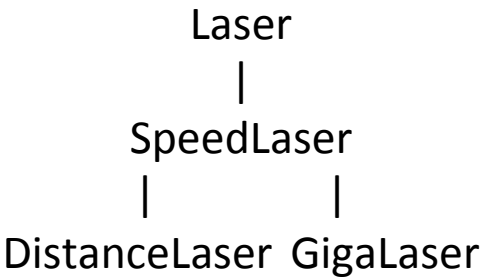
```
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override ShowID() = System.Console.Write(base.ID+”.v2”)
    member x.Accuracy = 80
type DistanceLaser() =
    inherit 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 ShowID() = System.Console.Write(base.ID+”.v2”)
    member x.Accuracy = 80
type DistanceLaser() =
    inherit SpeedLaser()
    abstract member Range : int
[<Sealed>]
type GigaLaser() =
    inherit SpeedLaser()
```

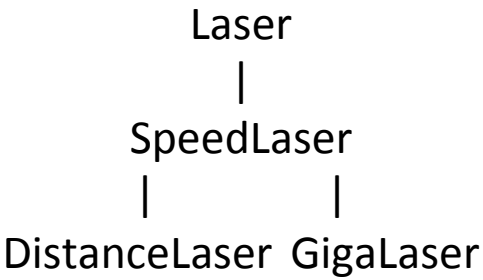
```
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override ShowID() = System.Console.Write(base.ID+”.v2”)
    member x.Accuracy = 80
type DistanceLaser() =
    inherit 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 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()
```

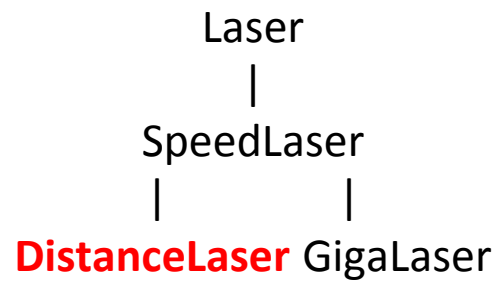


```
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override ShowID() = System.Console.Write(base.ID+”.v2”)
    member x.Accuracy = 80
type DistanceLaser() =
    inherit SpeedLaser()
    abstract member Range : int
[<Sealed>]
type GigaLaser() =
    inherit SpeedLaser()
let laser1 = GigaLaser()
laser1.ShowID()
```



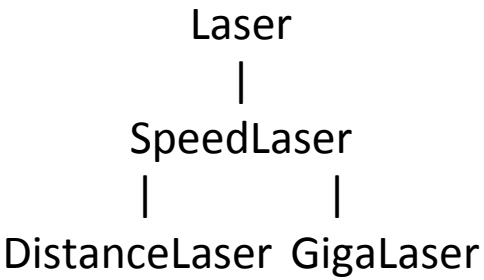
What does this output?

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

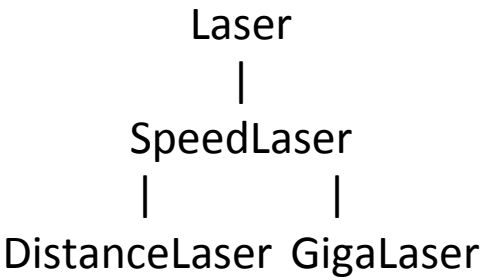


ERROR: “No implementation was given for 'abstract member DistanceLaser.Range : int'”

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

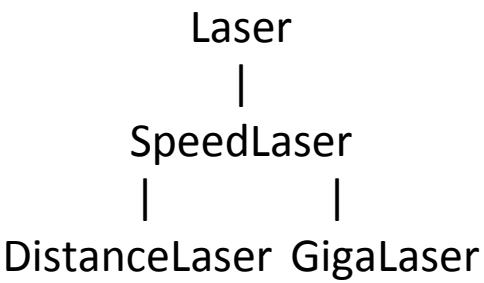


```
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override ShowID() = System.Console.Write(base.ID+”.v2”)
    member x.Accuracy = 80
type DistanceLaser() =
    inherit SpeedLaser()
    abstract member Range : int
[<Sealed>]
type GigaLaser() =
    inherit SpeedLaser()
let laser1 = GigaLaser()
let laser2 = { new DistanceLaser() with member x.Range = 10 }
laser1.ShowID()
```



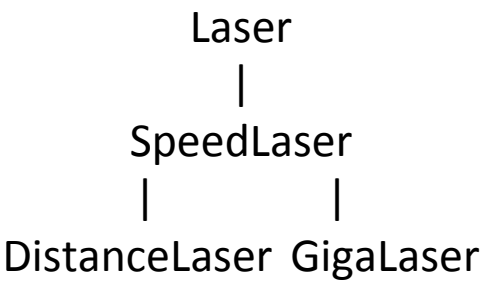
ERROR: “No implementation was given for 'abstract member DistanceLaser.Range : int'”

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

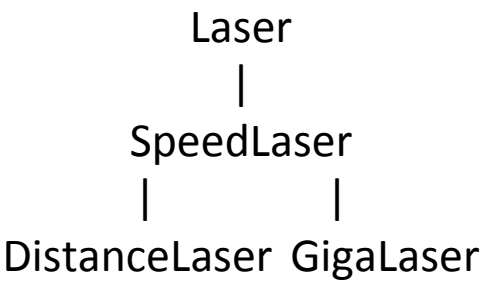


ERROR SHOULD BE: “you have an abstract class that you have not declared abstract'”

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

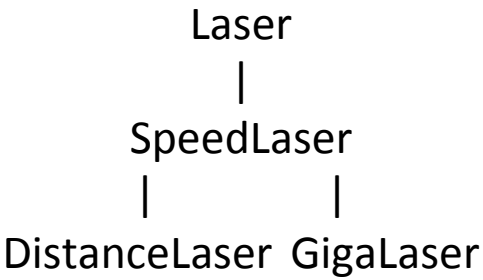


```
type Laser() =
    member x.ID = “Galaxy”
    abstract member ShowID : unit -> unit
    default x.ShowID() = System.Console.Write(x.ID)
type SpeedLaser() =
    inherit Laser()
    override ShowID() = System.Console.Write(base.ID+”.v2”)
    member x.Accuracy = 80
[<AbstractClass>]
type DistanceLaser() =
    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()
```



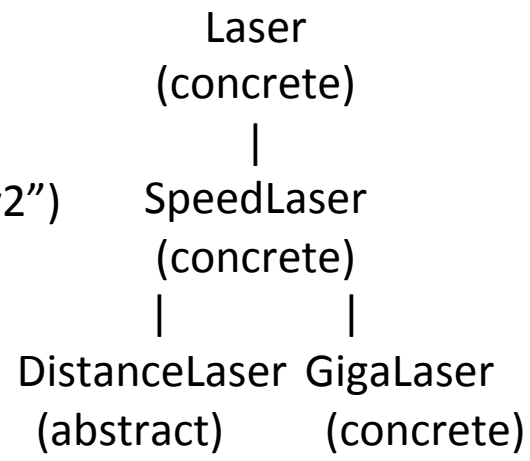
Galaxy

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



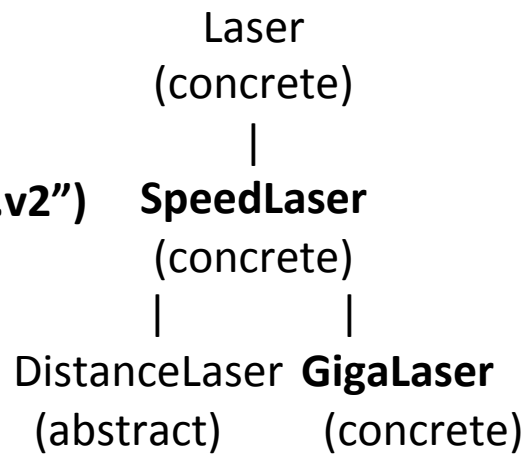
Galaxy

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



Galaxy

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



Galaxy

Recap today's lecture

- Class inheritance
  - Overriding
  - Abstract & concrete classes
  - Delegation
  - Sealed classes