Programmering og Problemløsning

10 December 2018

Christina Lioma

c.lioma@di.ku.dk

Today's lecture

- Method overloading
- Class inheritance

PSEUDOCODE

```
type Laser() = class
    Power = … remaining battery power
    Accuracy = … in finding target
```

PSEUDOCODE

```
type Laser() = class
    Power = … remaining battery power
    Accuracy = … in finding target

    Shoot() = … power decreases
    Scan() = … power decreases but accuracy increases
```

```
type Laser() =


    member x.Power =
    member x.Accuracy =
    member x.Shoot() =
    member x.Scan() =
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Power = power
    member x.Accuracy = accuracy
    member x.Shoot() =
    member x.Scan() =
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Power = power
    member x.Accuracy = accuracy
    member x.Shoot() =
    member x.Scan() =
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    ~~member x.Power = power~~
    ~~member x.Accuracy = accuracy~~
    member x.Shoot() =
    member x.Scan() =
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
```

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
let laser1 = new Laser(80.0, 60.0)
```

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
```

```
power: 72.000000, accuracy: 60.000000
```

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
laser1.Scan()
```

```
power: 72.000000, accuracy: 60.000000
power: 64.800000, accuracy: 66.00000
```

Test random power & accuracy values?

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
laser1.Scan()
```

```
power: 72.000000, accuracy: 60.000000
power: 64.800000, accuracy: 66.00000
```

System.Random(), Next()

```
let rnd = System.Random()
let rnd_nxt = rnd.Next()
```

https://msdn.microsoft.com/en-us/library/system.random

System.Random(), Next()

```
let rnd = System.Random()
let rnd_nxt = rnd.Next()
```

- *Next(): returns integer*
- *Next(max): returns integer up to but* ***excluding max***
- *Next(min, max): returns integer from* ***min (inclusive)*** *up to and* ***excluding max***
    - *1st value must be smaller or equal to 2nd value*

https://msdn.microsoft.com/en-us/library/system.random

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
```

Test random power & accuracy values?

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
```

Test random power & accuracy values?

```
type Laser() =
    let rnd = System.Random()
    let mutable power = float(rnd.Next(100))
    let mutable accuracy = float(rnd.Next(100))
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
```

```
type Laser() =
    let rnd = System.Random()
    let mutable power = float(rnd.Next(100))
    let mutable accuracy = float(rnd.Next(100))
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
let laser1 = new Laser()
laser1.Shoot()
laser1.Scan()
```

```
power: 18.000000, accuracy: 59.000000
power: 16.200000, accuracy: 64.90000
```

“Test random power & accuracy values”

“But also keep the option of specifying their values”

“Test random power & accuracy values”

Call the class without input arguments

Laser()

“But also keep the option of specifying their values”

“Test random power & accuracy values”

Call the class without input arguments

Laser()

“But also keep the option of specifying their values”

Call the class with input arguments

Laser(80.0, 60.0)

“Test random power & accuracy values”

Call the class without input arguments

Laser()

“Option to specify one value only”

Call the class with only one input argument

Laser(80.0)

“But also keep the option of specifying their values”

Call the class with input arguments

Laser(80.0, 60.0)

Method Overloading

“Test random power & accuracy values”

Call the class without input arguments

Laser()

“Option to specify one value only”

Call the class with only one input argument

Laser(80.0)

“But also keep the option of specifying their values”

Call the class with input arguments

Laser(80.0, 60.0)

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
```

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
```

Additional constructor

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
let laser1 = new Laser(80.0, 60.0)
```

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
let laser1 = new Laser(80.0, 60.0)
let laser2 = new Laser()
```

Additional constructor(s):

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
```

Additional constructor(s):

- *new() and indented body*

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
```

Additional constructor(s):

- *new() and indented body*
- *arguments, if any, between brackets*

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
```

Additional constructor(s):

- *new() and indented body*
- *arguments, if any, between brackets*
- *must always call the primary constructor*

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power * 0.9
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        power <- power * 0.9
        accuracy <-  accuracy * 1.1
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
```

Additional constructor(s):

- *new() and indented body*
- *arguments, if any, between brackets*
- *must always call the primary constructor (“new” is optional)*

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        if power > 50.0 then power <- power * 0.9 else power <- power * 0.7
        accuracy <-  accuracy * 1.05
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        ~~new~~ Laser(pow, acc)
```

Additional constructor(s):

- *new() and indented body*
- *arguments, if any, between brackets*
- *must always call the primary constructor*
- *let bindings. NO do bindings*

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        if power > 50.0 then power <- power * 0.9 else power <- power * 0.7
        accuracy <-  accuracy * 1.05
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
```

Additional constructor(s):

- *new() and indented body*
- *arguments, if any, between brackets*
- *must always call the primary constructor*
- *let bindings. NO do bindings*
- *then*

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        do printfn “power: %f, accuracy: %f” power accuracy
    member x.Scan() =
        if power > 50.0 then power <- power * 0.9 else power <- power * 0.7
        accuracy <-  accuracy * 1.05
        do printfn ”power: %f, accuracy: %f” power accuracy
    new() =
        let rnd = System.Random()
        let pow = float(rnd.Next(100))
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
        then printfn “Creating laser with random power & accuracy”
```

Method Overloading

When two or more methods in the same class have the exact *same* name but *different* parameters

Method Overloading

When two or more methods in the same class have the exact *same* name but *different* parameters

- Different number of parameters

Laser(80.0, 60.0)

Laser(80.0)

Laser()

Method Overloading

When two or more methods in the same class have the exact *same* name but *different* parameters

- Different number of parameters

  Laser(80.0, 60.0)

  Laser(80.0)

  Laser()

- Parameters of different data type

  Laser(80, 60)

  Laser(80.0, 60.0)

PSEUDOCODE

```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() = …
```

PSEUDOCODE

```
type Laser(p, a) =                      → Primary constructor
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() = …
    new(pow) =                          → Additional constructor 1
        acc generated randomly
        new Laser(pow, acc)
```

PSEUDOCODE

```
type Laser(p, a) =                          → Primary constructor
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() = …
    new(pow) =                              → Additional constructor 1
        acc generated randomly
        new Laser(pow, acc)
    new(acc) =                              → Additional constructor 2
        pow generated randomly
        new Laser(pow, acc)
```

PSEUDOCODE

```
type Laser(p, a) =                    → Primary constructor
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() = …
    new(pow) =                        → Additional constructor 1
        acc generated randomly
        new Laser(pow, acc)
    new(acc) =                        → Additional constructor 2
        pow generated randomly
        new Laser(pow, acc)
let laser1 = new Laser(60.0)
```

PSEUDOCODE

```
type Laser(p, a) =                    → Primary constructor
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() = …
    new(pow) =                        → Additional constructor 1
        acc generated randomly
        new Laser(pow, acc)
    new(acc) =                        → Additional constructor 2
        pow generated randomly
        new Laser(pow, acc)
let laser1 = new Laser(60.0)
```

Which constructor is used?

PSEUDOCODE

```
type Laser(p, a) =                        → Primary constructor
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() = …
    new(pow) =                            → Additional constructor 1
        acc generated randomly
        new Laser(pow, acc)
    new(acc) =                            → Additional constructor 2
        pow generated randomly
        new Laser(pow, acc)
let laser1 = new Laser(60.0)
```

Error: A unique overload for method 'Laser' could not be determined

Method Overloading

When two or more methods in the same class have the exact *same* name but *different* parameters

- Different number of parameters
- Parameters of different data type

It must be clear which constructor should be used → *code must determine unique overload*

- ~~Method overloading~~
- **Class inheritance**

```
type Laser(power, accuracy) = class
    Power = … remaining battery power
    Accuracy = … in finding target
    Shoot() = … power decreases
    Scan() =  … power decreases but accuracy increases
end
```

```
type Laser(power, accuracy) = class
        Power = … remaining battery power
        Accuracy = … in finding target
        Shoot() = … power decreases
        Scan() =  … power decreases but accuracy increases
end


type SpeedLaser(power, accuracy) = class
        Power = … remaining battery power
        Accuracy = … in finding target
        Shoot() = … power decreases
        Scan() =  … power decreases but accuracy increases
        SpeedShoot() = … shoots at tiny intervals
end
```

```
type Laser(power, accuracy) = class
    Power = … remaining battery power
    Accuracy = … in finding target
    Shoot() = … power decreases
    Scan() =  … power decreases but accuracy increases
end
```

identical

```
type SpeedLaser(power, accuracy) = class
    Power = … remaining battery power
    Accuracy = … in finding target
    Shoot() = … power decreases
    Scan() =  … power decreases but accuracy increases
    SpeedShoot() = … shoots at tiny intervals
end
```

```
type Laser(power, accuracy) = class
      Power = …
      Accuracy = …
      Shoot() = …
      Scan() =  …
end


type SpeedLaser (power, accuracy) = class
      inherit Laser(power, accuracy)
      SpeedShoot() = …
end
```

Inheritance

```
type Laser(power, accuracy) = class
        Power = …
        Accuracy = …
        Shoot() = …
        Scan() =  …
end
```

Base class

```
type SpeedLaser (power, accuracy) = class
        inherit Laser(power, accuracy)
        SpeedShoot() = …
end
```

Derived class from the base

Inheritance

Inheritance

BaseClass

attribute members

method members

DerivedClass

inherits **all** attribute & method members from Base

Inheritance

BaseClass (a.k.a. *Parent* or *Super* class)

attribute members

method members

DerivedClass (a.k.a. *Child* or *Sub* class)

inherits **all** attribute & method members from Base

Inheritance

BaseClass (a.k.a. *Parent* or *Super* class)

- attribute members
- method members

DerivedClass (a.k.a. *Child* or *Sub* class)

- inherits **all** attribute & method members from Base
- new attribute members
- new method members

Inheritance

BaseClass (a.k.a. *Parent* or *Super* class)

attribute members

method members

DerivedClass (a.k.a. *Child* or *Sub* class)

inherits **all** attribute & method members from Base

new attribute members

new method members

can add new attribute & method members in Derived, but

Base cannot access them

```
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 = Laser(90.0, 90.0)
let laser2 = SpeedLaser(100.0, 100.0)
```

```
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 = Laser(90.0, 90.0)
let laser2 = SpeedLaser(100.0, 100.0)
laser1.Shoot()
laser2.Shoot()
```

Output?

```
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 = Laser(90.0, 90.0)
let laser2 = SpeedLaser(100.0, 100.0)
laser1.Shoot()
laser2.Shoot()
```

```
Power left: 89.000000
Power left: 99.000000
```

```
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)
      member x.SpeedShoot() =
            power <- power – 10.0
            printfn “Power left: %f” power
```

Add new method to Derived class

```
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)
    member x.SpeedShoot() =
        power <- power – 10.0
        printfn “Power left: %f” power

let laser2 = SpeedLaser(100.0, 100.0)
laser2.SpeedShoot()
```

Add new method to Derived class

Output?

```
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)
    member x.SpeedShoot() =
        power <- power – 10.0
        printfn “Power left: %f” power

let laser2 = SpeedLaser(100.0, 100.0)
laser2.SpeedShoot()
```

Error: ‘power’ is not defined

```
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)
    member x.SpeedShoot() =
        power <- power – 10.0
        printfn “Power left: %f” power

let laser2 = SpeedLaser(100.0, 100.0)
laser2.SpeedShoot()
```

‘power’ is not inherited

Error: ‘power’ is not defined

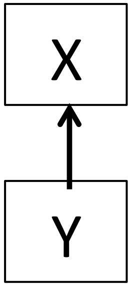
```
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    member x.Shoot() =
        power <- power – 1.0
        printfn “Power left: %f” power

type SpeedLaser(p, a) =
    inherit Laser(p, a)
    let mutable power = p
    member x.SpeedShoot() =
        power <- power – 10.0
        printfn “Power left: %f” power

let laser2 = SpeedLaser(100.0, 100.0)
laser2.SpeedShoot()
```

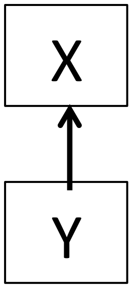
```
Power left: 90.000000
```

Inheritance types

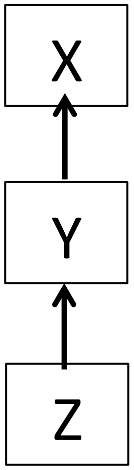


Single

Inheritance types

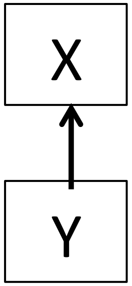


Single



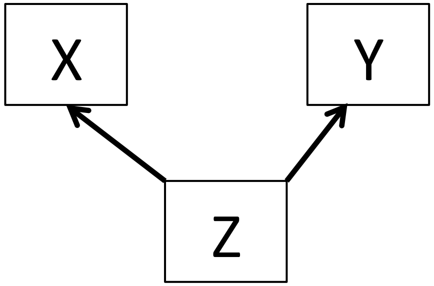
Multi-level

Inheritance types



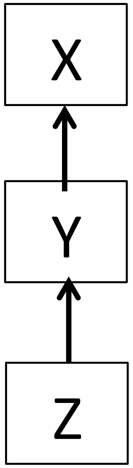
Single

F# allows



Multiple

F# does not allow



Multi-level

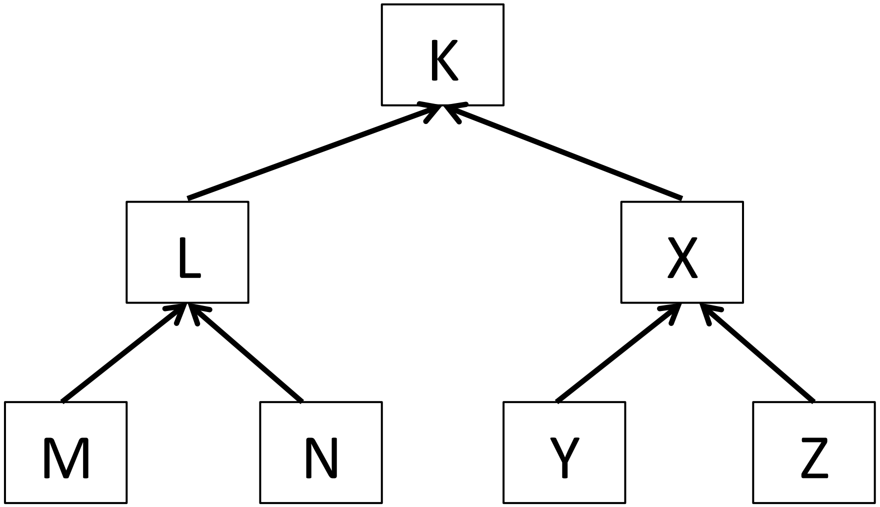
F# allows

Inheritance

- Code reusability (inherited class members)
- Code extensibility (new Derived class members extend Base)

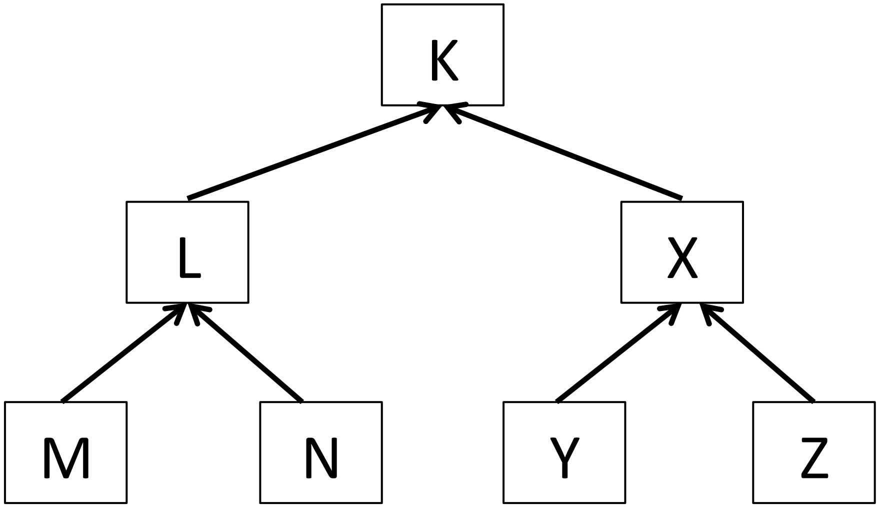
Inheritance

- Code reusability (inherited class members)
- Code extensibility (new Derived class members extend Base)
- If Base changes, all its Derived classes are affected



Inheritance

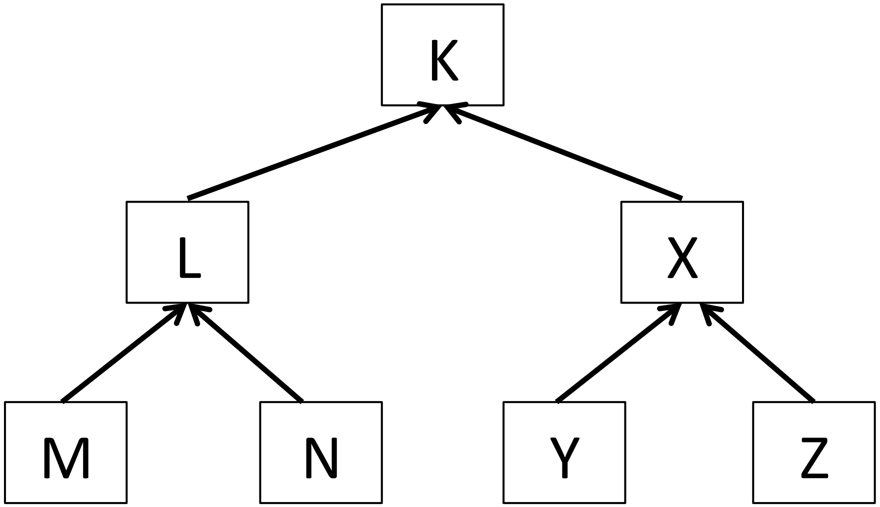
- Code reusability (inherited class members)
- Code extensibility (new Derived class members extend Base)
- If Base changes, all its Derived classes are affected



- In large class hierarchy, several class members remain unused even though memory is allocated to them

Inheritance

- Code reusability (inherited class members)
- Code extensibility (new Derived class members extend Base)
- If Base changes, all its Derived classes are affected



- In large class hierarchy, several class members remain unused even though memory is allocated to them
- If no base class is specified, we implicitly inherit from System.Object

What happens with inheritance to…

- *Default (or instance) vs static members?*

- *Member accessibility with get() and set()?*

- *Empty classes?*

- *Additional constructors?*

What happens with inheritance to…

- *Default (or instance) vs static members? → inherited without problems*

- *Member accessibility with get() and set()? → inherited without problems*

- *Empty classes? → inherited without problems*

- *Additional constructors? → special inheritance case*

```
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 = Laser(100, 100)
laser1.Shoot()
let laser2 = SpeedLaser(200, 200)
laser2.Shoot()
```

If BaseClass has additional constructors, are they inherited?

If BaseClass has additional constructors, are they inherited?

```
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 = Laser(100, 100)
laser1.Shoot()
let laser2 = SpeedLaser(200, 200)
laser2.Shoot()
```

Line 12: “A unique overload for method 'Laser' could not be determined based on type information prior to this program point.”

If BaseClass has additional constructors, are they inherited?

```
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)             ← ERROR OCCURS HERE
let laser1 = Laser(100, 100)
laser1.Shoot()
let laser2 = SpeedLaser(200, 200)
laser2.Shoot()
```

Line 12: “A unique overload for method 'Laser' could not be determined based on type information prior to this program point.”

OK, I will run both instances with floats (primary constructor)

```
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 = Laser(100.0, 100.0)          ← CALLING PRIMARY CONSTRUCTOR
laser1.Shoot()
let laser2 = SpeedLaser(200.0, 200.0)     ← CALLING PRIMARY CONSTRUCTOR
laser2.Shoot()
```

OK, I will run both instances with floats (primary constructor)

```
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 = Laser(100.0, 100.0)           ← CALLING PRIMARY CONSTRUCTOR
laser1.Shoot()
let laser2 = SpeedLaser(200.0, 200.0)      ← CALLING PRIMARY CONSTRUCTOR
laser2.Shoot()
```

Line 12: “A unique overload for method 'Laser' could not be determined based on type information prior to this program point.”

OK, I will skip the inherited instance altogether

```
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 = Laser(100.0, 100.0)           ← CALLING PRIMARY CONSTRUCTOR
laser1.Shoot()
~~let laser2 = SpeedLaser(200.0, 200.0)~~
~~laser2.Shoot()~~
```

OK, I will skip the inherited instance altogether

Now I cannot even run Base!

```
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 = Laser(100.0, 100.0)          ← CALLING PRIMARY CONSTRUCTOR
laser1.Shoot()
~~let laser2 = SpeedLaser(200.0, 200.0)~~
~~laser2.Shoot()~~
```

Line 12: “A unique overload for method 'Laser' could not be determined based on type information prior to this program point.”

If Base has additional constructor(s), must specify which constructor is inherited

```
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 : float, a : float)       ← SPECIFY INHERITED CONSTRUCTOR
let laser1 = Laser(100.0, 100.0)
laser1.Shoot()
~~let laser2 = SpeedLaser(200.0, 200.0)~~
~~laser2.Shoot()~~
```

Multiple constructor inheritance

Possible to inherit more than one constructor using F#’s explicit syntax:

https://msdn.microsoft.com/en-us/library/dd233225.aspx

Recap today’s lecture

- Class constructors (method overloading)
- Random(), Next()
- Related classes (inheritance)