# Programmering og Problemløsning

12 December 2016

Christina Lioma

c.lioma@di.ku.dk

# Today's lecture

- Class construction
- Random(), Next()
- Inheritance

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount
        with get() = count
    member x.Fire() = printfn "%s is firing" x.Name
end
printfn “Laser count: %i” Laser.LaserCount
```

```
Laser count: 0
```

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount
        with get() = count
    member x.Fire() = printfn "%s is firing" x.Name
end
printfn “Laser count: %i” Laser.LaserCount
```

***Why do we need get()?***

```
Laser count: 0
```

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount  = count
        ~~with get() = count~~
    member x.Fire() = printfn "%s is firing" x.Name
end
printfn “Laser count: %i” Laser.LaserCount
```

***Why do we need get()?***

***We do not***

```
Laser count: 0
```

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount  = count
        ~~with get() = count~~
    member x.Fire() = printfn "%s is firing" x.Name
end
printfn “Laser count: %i” Laser.LaserCount
Laser.LaserCount <- 100
printfn “Laser count: %i” Laser.LaserCount
```

***Why do we need get()?***
***We do not, unless we want***
***to change count’s value***

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount  = count
        ~~with get() = count~~
    member x.Fire() = printfn "%s is firing" x.Name
end
printfn “Laser count: %i” Laser.LaserCount
Laser.LaserCount <- 100
printfn “Laser count: %i” Laser.LaserCount
```

***Why do we need get()?***
***We do not, unless we want to change count’s value***

***Property 'LaserCount' cannot be set***

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount
        ~~with get() = count~~
        with set(value) = count <- value
    member x.Fire() = printfn "%s is firing" x.Name
end
printfn “Laser count: %i” Laser.LaserCount
Laser.LaserCount <- 100
printfn “Laser count: %i” Laser.LaserCount
```

***Why do we need get()?***
***We do not, unless we want***
***to change count’s value***

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount
        ~~with get() = count~~
        with set(value) = count <- value
    member x.Fire() = printfn "%s is firing" x.Name
end
printfn “Laser count: %i” Laser.LaserCount
Laser.LaserCount <- 100
printfn “Laser count: %i” Laser.LaserCount
```

***Why do we need get()?***
***We do not, unless we want to change count’s value***

***Property 'LaserCount' is not readable***

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount
        ~~with get() = count~~
        with set(value) = count <- value
    member x.Fire() = printfn "%s is firing" x.Name
end
~~printfn “Laser count: %i” Laser.LaserCount~~
Laser.LaserCount <- 100
~~printfn “Laser count: %i” Laser.LaserCount~~
```

***Why do we need get()?***
***We do not, unless we want***
***to change count’s value***

***This runs ok, but does not output anything***

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount
        with get() = count
        and set(value) = count <- value
    member x.Fire() = printfn "%s is firing" x.Name
end
printfn “Laser count: %i” Laser.LaserCount
Laser.LaserCount <- 100
printfn “Laser count: %i” Laser.LaserCount
```

***Why do we need get()?***
***We do not, unless we want to change count’s value***

```
Laser count: 0
Laser count: 100
```

```
type Laser(name) = class
    static let mutable count = 0
    do
        count <- count + 1
    member x.Name = name
    static member LaserCount
        with get() = count
        and set(value) = count <- value
    member x.Fire() = printfn "%s is firing" x.Name
end
printfn “Laser count: %i” Laser.LaserCount
Laser.LaserCount <- 100
printfn “Laser count: %i” Laser.LaserCount
```

***If we do not change count’s value, we can print it without get().***

***If we change it, we need get() to print it***

*Laser count: 0*
*Laser count: 100*

```
type Laser(name) = class
    member x.Name = name
    member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser("Super Laser”)
laser1.Fire()
laser1.Fire()
laser1.Fire()
```

```
type Laser(name) = class
    member x.Name = name
    member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser("Super Laser”)
laser1.Fire()
laser1.Fire()
laser1.Fire()
```

**Generate a random laser ID every time I use the laser instance**

```
type Laser(name) = class
        member x.Name = name
        member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser("Super Laser”)
laser1.Fire()
laser1.Fire()
laser1.Fire()
```

## Generate a random laser ID every time I use the laser instance

```
let gen = System.Random()
let ran_int = gen.Next()
```

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

```
type Laser(name) = class
    let mutable laserID  = new System.Random()
    member x.MyLaserID = laserID.Next()
    member x.Name = name
    member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser(“Super Laser”)
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.MyLaserID
```

```
type Laser(name) = class
    let mutable laserID  = new System.Random()
    member x.MyLaserID = laserID.Next()
    member x.Name = name
    member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser(“Super Laser”)
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.MyLaserID
```

```
Laser ID: 1415622264
Laser ID: 2110008525
```

```
type Laser(name) = class
    let mutable laserID  = new System.Random()
    member x.MyLaserID = laserID.Next()
    member x.Name = name
    member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser(“Super Laser”)
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.MyLaserID
```

***Did you try defining member with val?***

```
Laser ID: 1415622264
Laser ID: 2110008525
```

```
type Laser(name) = class
    let mutable laserID  = new System.Random()
    member x.MyLaserID = laserID.Next()
    member val YourLaserID = laserID.Next() with get
    member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser(“Super Laser”)
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.YourLaserID
printfn ”Laser ID: %i" laser1.YourLaserID
```

```
type Laser(name) = class
        let mutable laserID  = new System.Random()
        member x.MyLaserID = laserID.Next()
        member val YourLaserID = laserID.Next() with get
        member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser(“Super Laser”)
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.YourLaserID
printfn ”Laser ID: %i" laser1.YourLaserID
```

```
type Laser(name) = class
    let mutable laserID  = new System.Random()
    member x.MyLaserID = laserID.Next()
    member val YourLaserID = laserID.Next() with get
    member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser(“Super Laser”)
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.YourLaserID
printfn ”Laser ID: %i" laser1.YourLaserID
```

```
Laser ID: 1021175711
Laser ID: 141436640
Laser ID: 1668647548
Laser ID: 1668647548
```

```
type Laser(name) = class
    let mutable laserID  = new System.Random()
    member x.MyLaserID = laserID.Next()
    member val YourLaserID = laserID.Next() with get
    member x.Fire() = printfn "%s is firing" x.Name
end
let laser1 = new Laser(“Super Laser”)
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.MyLaserID
printfn ”Laser ID: %i" laser1.YourLaserID
printfn ”Laser ID: %i" laser1.YourLaserID
```

```
Laser ID: 1021175711
Laser ID: 141436640
Laser ID: 1668647548
Laser ID: 1668647548
```

***MyLaserID changes when called repeatedly***

***YourLaserID does not change when called repeatedly***

F# Design Choice:

```
member x.MyLaserID = laserID.Next()
```

- Evaluated every time MyLaserID is accessed

```
member val YourLaserID = laserID.Next() with get, set
```

- Only evaluated once when YourLaserID is initialised

Read more: https://msdn.microsoft.com/en-us/library/dd483467.aspx

```
type Laser() =
    Power = …
    Accuracy = …
    Shoot() = …
```

```
type Laser() =
    Power = … remaining battery power (measured in some unit)
    Accuracy = … in finding target (measured in some unit)
    Shoot() = … power decreases per shot
```

```
type Laser() =
    Power = … remaining battery power (measured in some unit)
    Accuracy = … in finding target (measured in some unit)
    Shoot() = … power decreases per shot
    Scan() = … power decreases but accuracy increases
```

```
type Laser() =
    Power = … remaining battery power (measured in some unit)
    Accuracy = … in finding target (measured in some unit)
    Shoot() = … power decreases per shot
    Scan() = … power decreases but accuracy increases
                when power > 50 units
                    power decreases slowly
                otherwise
                    power decreases quickly
```

```
type Laser() =
    Power = … remaining battery power (measured in some unit)
    Accuracy = … in finding target (measured in some unit)
    Shoot() = … power decreases per shot
    Scan() = … power decreases but accuracy increases
                when power > 50 units
                    power decreases slowly (at rate x)
                otherwise
                    power decreases quickly (at rate y, where y > x)
```

```
type Laser() =
    Power = … remaining battery power (measured in some unit)
    Accuracy = … in finding target (measured in some unit)
    Shoot() = … power decreases per shot
    Scan() = … power decreases but accuracy increases
                  when power > 50 units
                      power decreases slowly (at rate x)
                  otherwise
                      power decreases quickly (at rate y, where y > x)
                  accuracy increases always at the same rate
```

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


    member x.Scan() =
type Laser(p, a) =
    let mutable power = p
    let mutable accuracy = a
    ~~member x.Accuracy = accuracy~~
    ~~member x.Power = power~~
    member x.Shoot() =


    member x.Scan() =
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() =
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
```

```
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
```

```
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
let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
laser1.Scan()
```

```
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
let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
laser1.Scan()
```

```
power: 79.000000, accuracy: 60.000000
power: 71.100000, accuracy: 63.00000
```

***Test random power & accuracy values?***

```
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
let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
laser1.Scan()
```

```
power: 79.000000, accuracy: 60.000000
power: 71.100000, accuracy: 63.00000
```

***Test random power & accuracy values?***

***But also keep the option of specifying their values?***

```
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
let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
laser1.Scan()
```

```
power: 79.000000, accuracy: 60.000000
power: 71.100000, accuracy: 63.00000
```

***“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 – 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




let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
laser1.Scan()
```

```
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 qccuracy
    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)
let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
laser1.Scan()
```

***Additional constructor***

```
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)
let laser1 = new Laser(80.0, 60.0)
laser1.Shoot()
laser1.Scan()
```

```
power: 79.000000, accuracy: 60.000000
power: 71.100000, accuracy: 63.000000
```

```
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)
let laser2 = new Laser()
laser2.Shoot()
laser2.Scan()
```

```
power: 5.000000, accuracy: 79.000000
power: 3.500000, accuracy: 82.950000
```

***Call Laser() with one input only?***

```
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)
let laser2 = new Laser()
laser2.Shoot()
laser2.Scan()
```

```
power: 5.000000, accuracy: 79.000000
power: 3.500000, accuracy: 82.950000
```

```
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(pow) =
        let rnd = System.Random()
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
let laser2 = new Laser(80.0)
laser2.Shoot()
laser2.Scan()
```

***Call Laser() with one input only?***

***Call Laser() with one input only?***

```
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(pow) =
        let rnd = System.Random()
        let acc = float(rnd.Next(100))
        new Laser(pow, acc)
let laser2 = new Laser(80.0)
laser2.Shoot()
laser2.Scan()
```

```
power: 79.000000, accuracy: 8.000000
power: 71.100000, accuracy: 8.400000
```

## *Additional constructor(s):*

```
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”
```

***Additional constructor(s):***

- *new() and indented body*

```
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”
```

***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 – 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”
```

***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 – 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”
```

***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)
        then printfn “Creating laser with random power & accuracy”
```

***Additional constructor(s):***

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

```
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”
```

***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)

```
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(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        new Laser(floatP, floatA)
```

```
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(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        new Laser(floatP, floatA)
let laser3 = new Laser(50, 70)
laser1.Shoot()
laser1.Scan()
```

```
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(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        new Laser(floatP, floatA)
let laser3 = new Laser(50, 70)
laser1.Shoot()
laser1.Scan()
```

***What is the output’s data type?***

```
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(p : int, a : int) =
        let floatP = float(p)
        let floatA= float(a)
        new Laser(floatP, floatA)
let laser3 = new Laser(50, 70)
laser1.Shoot()
laser1.Scan()
```

```
power: 49.000000, accuracy: 70.000000
power: 34.300000, accuracy: 73.500000
```

```
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
```

***Base class***

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

***Derived class***
***from the base***

# Inheritance

# Inheritance

BaseClass

attributes

methods

DerivedClass

inherits **all** attributes & methods from Base

# Inheritance

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

attributes

methods

DerivedClass (a.k.a. *Child* or *Sub* class)
inherits **all** attributes & methods from Base

# Inheritance

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

- attributes
- methods

DerivedClass (a.k.a. *Child* or *Sub* class)
inherits **all** attributes & methods from Base

- newAttributes
- newMethods
- can add new attributes & methods in Derived, but
- Base cannot access them

# Recap today’s lecture

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