

Objects Comparisons

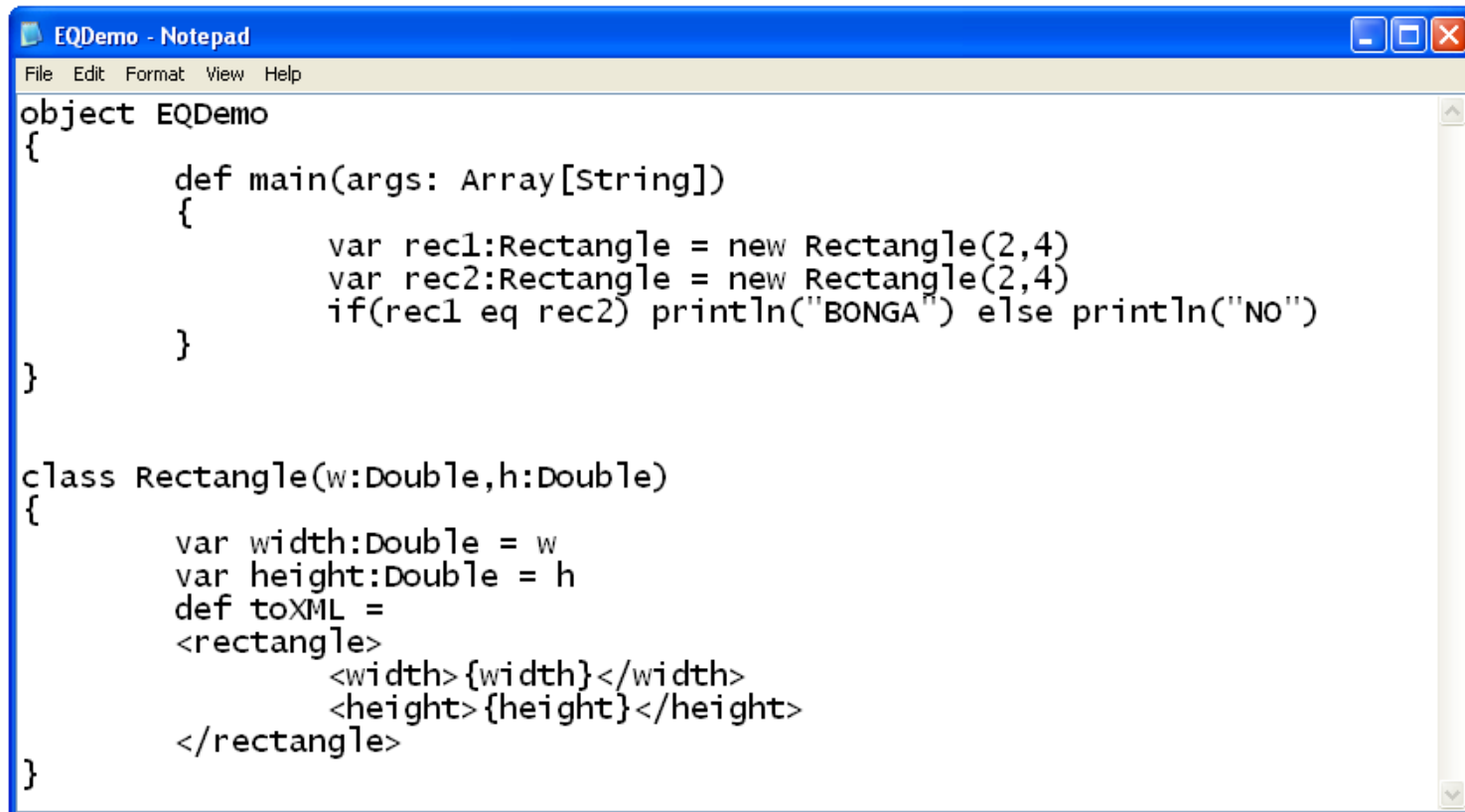
Introduction

- ❖ The Scala programming language support for equality is different comparing with the one we know in Java.

The `eq` Operator

- ❖ Comparing two objects using the `eq` operator is true if the two references are the same. They refer the same object.

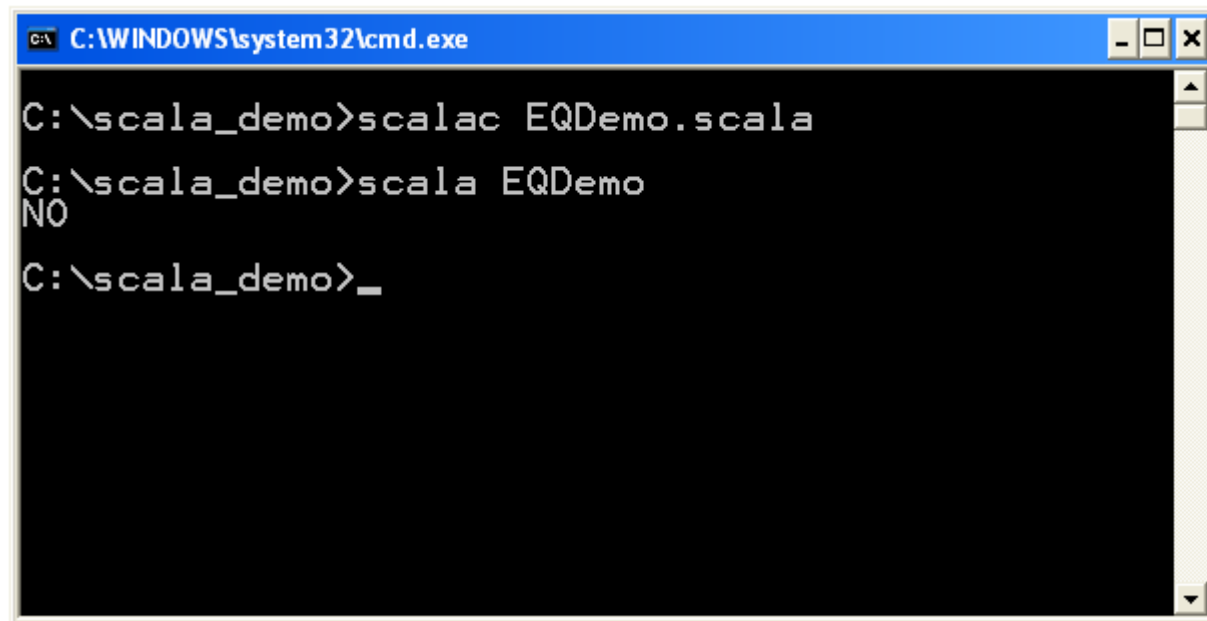
The eq Operator



```
EQDemo - Notepad
File Edit Format View Help
object EQDemo
{
    def main(args: Array[String])
    {
        var rec1:Rectangle = new Rectangle(2,4)
        var rec2:Rectangle = new Rectangle(2,4)
        if(rec1 eq rec2) println("BONGA") else println("NO")
    }
}

class Rectangle(w:Double,h:Double)
{
    var width:Double = w
    var height:Double = h
    def toXML =
    <rectangle>
        <width>{width}</width>
        <height>{height}</height>
    </rectangle>
}
```

The eq Operator



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scalac EQDemo.scala
C:\scala_demo>scala EQDemo
NO
C:\scala_demo>_
```

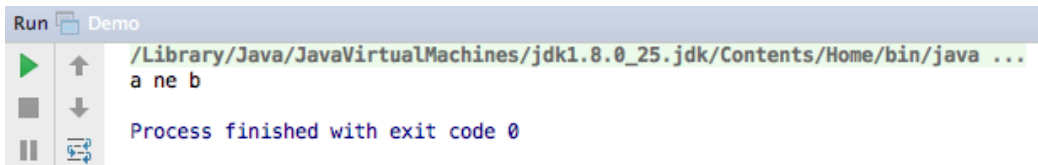
The `ne` Operator

- ❖ Comparing two objects using the `ne` operator is true if the two references are different. They refer different objects.

The ne Operator

```
object Demo {  
  def main(args:Array[String]):Unit = {  
    val a = Rectangle(3,4)  
    val b = Rectangle(3,4)  
    if(a ne b) println("a ne b")  
  }  
}
```

```
case class Rectangle(private var width:Double,private var height:Double)
```



```
Run Demo  
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...  
a ne b  
Process finished with exit code 0
```

The `equals` Method

- ❖ Comparing two objects using the `equals` method inherited from `Object` returns `true` if the two references refer to the very same object.
- ❖ Overriding this method we should override the `hashCode` method as well. Otherwise, we might get into unexpected behavior when working with collections such as `HashSet`.

The equals Method

```
package com.abelski.samples

import java.io.PrintWriter;
import java.io.File;

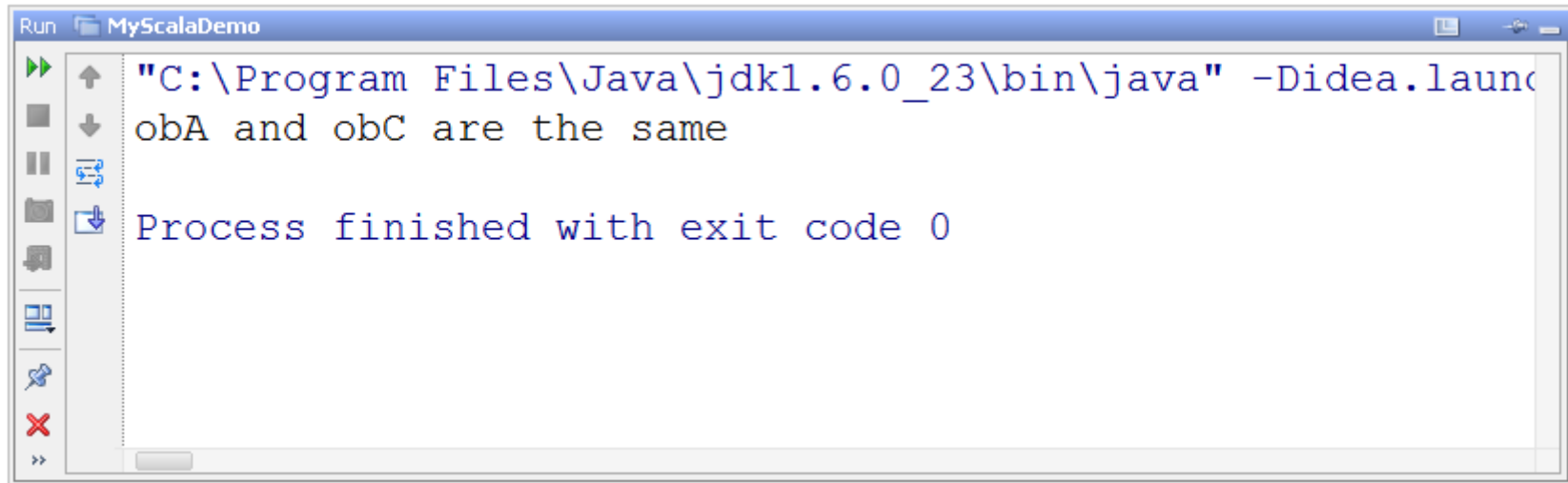
object MyScalaDemo extends Application
{
    val obA = new SportiveWeight(8)
    val obB = new SportiveWeight(10)
    val obC = new SportiveWeight(8)
    if(obA.equals(obB))
    {
        println("obA and obB are the same")
    }
    if(obA.equals(obC))
    {
        println("obA and obC are the same")
    }
}
```



The equals Method

```
class SportiveWeight(number:Int)
{
  private var weightvalue:Int = if(number>0)number else 10
  def weight_=(Int:Double)
  {
    weightvalue = number
  }
  def weight = weightvalue
  override def equals(ob:Any):Boolean =
  {
    ob match
    {
      case ob:SportiveWeight => this.weight==ob.weight
      case _ => false
    }
  }
  override def hashCode = weightvalue
}
```

The equals Method



```
Run MyScalaDemo
>> "C:\Program Files\Java\jdk1.6.0_23\bin\java" -Didea.launch
obA and obC are the same
Process finished with exit code 0
```

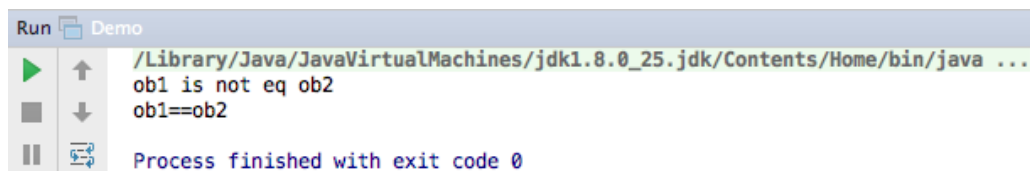
Case Classes

- ❖ When defining a case class we will get an automatic new implementation for the `equals` method. Other methods we automatically get their implementation include the `toString` and `hashCode`.

Case Classes

```
object Demo {  
  def main(args:Array[String]):Unit = {  
  
    var ob1 = Rectangle(3,4)  
    var ob2 = Rectangle(3,4)  
    if(ob1 eq ob2) println("ob1 eq ob2") else println("ob1 is not eq ob2")  
    if(ob1==ob2) println("ob1==ob2") else println("ob1 is not == ob2")  
  
  }  
}
```

```
case class Rectangle(private var width:Double,private var height:Double)
```



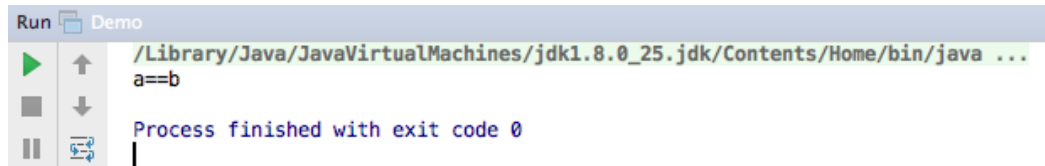
The screenshot shows a run console window titled "Run Demo". The command executed is `/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...`. The output consists of two lines: `ob1 is not eq ob2` and `ob1==ob2`. The process finished with exit code 0.

Comparing Strings

- ❖ When comparing two strings using the `==` operator it is the same as comparing two different objects from the same class. Indirectly the `equals` method is invoked.
- ❖ For that reason, comparing strings in Scala is different comparing with what we know in Java.

Comparing Strings

```
object Demo {  
  def main(args:Array[String]):Unit = {  
    var a:String = "abcefg hij".substring(2)  
    var b:String = "abcefg hij".substring(2)  
    if(a==b) println("a==b")  
  }  
}
```



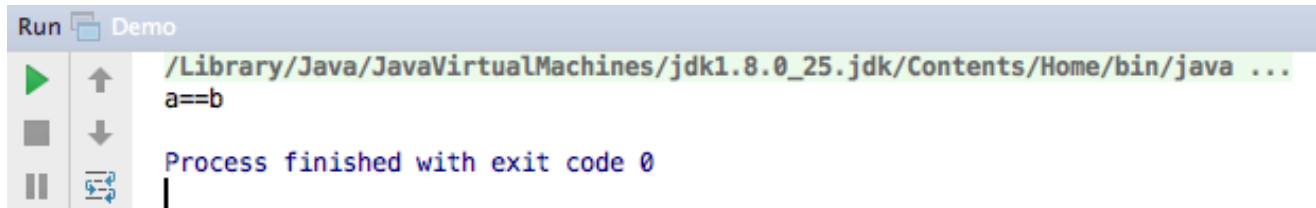
```
Run Demo  
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...  
a==b  
Process finished with exit code 0
```

Comparing Value Type Values

- ❖ When comparing two value type values using the `==` operator it is the same as with comparing any two objects.
- ❖ When the two value type values are of different types, before the comparison takes place new object will be created in order to have a comparison of two objects of the same value type.

Comparing Value Type Values

```
object Demo {  
  def main(args:Array[String]):Unit = {  
    val a:Int = 12  
    val b:Double = 12.0  
    if(a==b) println("a==b")  
  }  
}
```



```
Run Demo  
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...  
a==b  
Process finished with exit code 0
```

The hashCode Function

- ❖ When overriding the `equals` method we must also override the `hashCode` method.
- ❖ If two objects are equal according to the `equals` method then calling the `hashCode` method on each one of the two objects should return the same integer value.
- ❖ If we avoid this rule then collections as `Set` and `Map` will not work as expected.

The hashCode Function

```
class Point(val x:Int, val y:Int) {  
  override def hashCode = 12 * (12+x) + y  
  override def equals(other:Any):Boolean = {  
    other match  
    {  
      case other: Point => this.x == other.x && this.y == other.y  
      case _ => false  
    }  
  }  
}
```

Make sure the other parameter of the equals method is of the Any type. Otherwise, it won't be overriding. It will be overloading.

The hashCode Function

```
object Demo {  
  def main(args:Array[String]):Unit = {  
    val a = new Point(3,4)  
    val b = new Point(3,4)  
    if(a equals b) println("a equal b")  
  }  
}
```



Run Demo

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...  
a equal b  
Process finished with exit code 0
```

The hashCode Function

- ❖ When defining our class as a `case` class then both the `hashCode` and the `equals` methods are automatically defined for us.

The == Operator

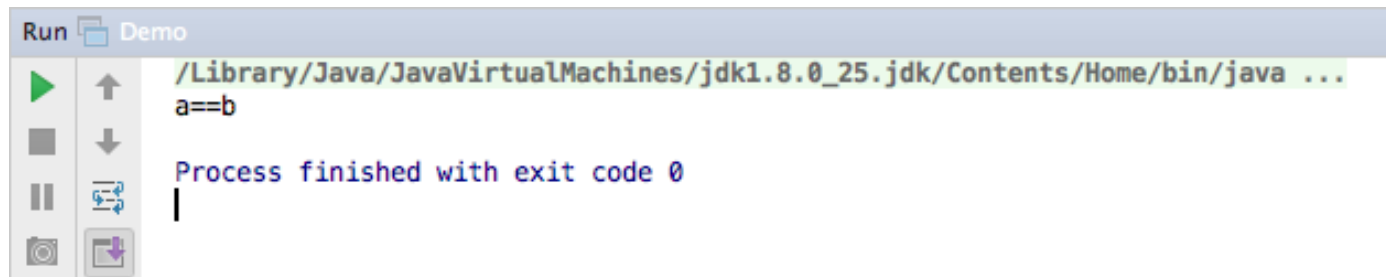
- ❖ Comparing two objects using the == operator we will get indirect invocation for the `equals` method.
- ❖ We can override the `equals` method and by doing so influence the way the == operator works.

The == Operator

```
class Point(val x:Int, val y:Int) {  
  override def hashCode = 12 * (12+x) + y  
  override def equals(other:Any):Boolean = {  
    other match  
    {  
      case other: Point => this.x == other.x && this.y == other.y  
      case _ => false  
    }  
  }  
}
```

The == Operator

```
object Demo {  
  def main(args:Array[String]):Unit = {  
    val a = new Point(3,4)  
    val b = new Point(3,4)  
    if(a==b) println("a==b")  
  }  
}
```



```
Run Demo  
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...  
a==b  
Process finished with exit code 0
```