Scala

IntelliJ, packages, methods, types, conditionals

Hello World

Scala Basics

```scala
package lo1_program_execution.scala

object Hello {

  def main(args: Array[String]): Unit = {
    println("Hello Scala!")
  }

}
```

A first example in Scala

Prints "Hello Scala!" to the screen

Scala Basics

```scala
package lo1_program_execution.scala

object Hello {

  def main(args: Array[String]): Unit = {
    println("Hello Scala!")
  }

}
```

Package declaration

Should match the directory structure in the src directory

This file is saved in the directory "src/lo1_program_execution/scala"

To create a package, right click the src directory > new > package

Scala Basics

```scala
package lo1_program_execution.scala

object Hello {

  def main(args: Array[String]): Unit = {
    println("Hello Scala!")
  }

}
```

Objects

Objects can store variables and functions*

Name should match the filename

This code is in the file "src/lo1_program_execution/scala/Hello.scala"

*We call them methods when they are part of an object

Scala Basics

```scala
package lo1_program_execution.scala

object Hello {

  def main(args: Array[String]): Unit = {
    println("Hello Scala!")
  }

}
```

Main Method

The method that executes when you run your object/program

Will always have the header "**def** main(args: Array[String]): Unit"

Scala Basics

```scala
package lo1_program_execution.scala

object Hello {

  def main(args: Array[String]): Unit = {
    println("Hello Scala!")
  }

}
```

Print Line

Prints "Hello Scala!" to the screen

Methods
Variables

```scala
package lo1_program_execution.scala

object FirstObject {

  def multiplyByTwo(input: Double): Double = {
    input * 2.0
  }

  def main(args: Array[String]): Unit = {
    var x: Double = 7.0
    var result = multiplyByTwo(x)
    println(result)
  }

}
```

Prints 14.0 to the screen

```scala
package lo1_program_execution.scala

object FirstObject {

  def multiplyByTwo(input: Double): Double = {
    input * 2.0
  }

  def main(args: Array[String]): Unit = {
    var x: Double = 7.0
    var result = multiplyByTwo(x)
    println(result)
  }

}
```

Methods

Must declare types!

This method takes a Double as a parameter and returns a Double

```scala
package lo1_program_execution.scala

object FirstObject {

  def multiplyByTwo(input: Double): Double = {
    input * 2.0
  }

  def main(args: Array[String]): Unit = {
    var x: Double = 7.0
    var result = multiplyByTwo(x)
    println(result)
  }

}
```

Methods

Must explicitly define the **type** of each parameter

Parameter name and type are separated by a colon :

```scala
package lo1_program_execution.scala

object FirstObject {

  def multiplyByTwo(input: Double): Double = {
    input * 2.0
  }

  def main(args: Array[String]): Unit = {
    var x: Double = 7.0
    var result = multiplyByTwo(x)
    println(result)
  }

}
```

Methods

The return **type** must also be explicit

Return type follows the parameter list

Parameter list and return type are separated by a colon :

```scala
package lo1_program_execution.scala

object FirstObject {

  def multiplyByTwo(input: Double): Double = {
    input * 2.0
  }

  def main(args: Array[String]): Unit = {
    var x: Double = 7.0
    var result = multiplyByTwo(x)
    println(result)
  }

}
```

Methods

The body of the method is executed when the method is called

Return statements are optional (And discouraged)

The returned value is the last expression that's evaluated during the method call

```scala
package lo1_program_execution.scala

object FirstObject {

  def multiplyByTwo(input: Double): Double = {
    input * 2.0
  }

  def main(args: Array[String]): Unit = {
    var x: Double = 7.0
    var result = multiplyByTwo(x)
    println(result)
  }

}
```

Variables

Type declaration [sometimes] optional, but helpful

Creates a mutable* variable named **x** of type Double and initializes it to the value 7.0

*Value can change

```scala
package lo1_program_execution.scala

object FirstObject {

  def multiplyByTwo(input: Double): Double = {
    input * 2.0
  }

  def main(args: Array[String]): Unit = {
    var x: Double = 7.0
    var result = multiplyByTwo(x)
    println(result)
  }

}
```

Variables

Variable declaration without a type

Type is inferred by the return type of the method

Conditionals

```scala
package lo1_program_execution.scala

object Conditional {

  def computeSize(input: Double): String = {
    val large: Double = 60.0
    val medium: Double = 30.0
    if (input >= large) {
      "large"
    } else if (input >= medium) {
      "medium"
    } else {
      "small"
    }
  }

  def main(args: Array[String]): Unit = {
    println(computeSize(70.0))
    println(computeSize(50.0))
    println(computeSize(10.0))
  }

}
```

Prints:
large
medium
small

```scala
package lo1_program_execution.scala

object Conditional {

  def computeSize(input: Double): String = {
    val large: Double = 60.0
    val medium: Double = 30.0
    if (input >= large) {
      "large"
    } else if (input >= medium) {
      "medium"
    } else {
      "small"
    }
  }

  def main(args: Array[String]): Unit = {
    println(computeSize(70.0))
    println(computeSize(50.0))
    println(computeSize(10.0))
  }

}
```

Values declared with **val** cannot change

Reassignment causes an error

```scala
package lo1_program_execution.scala

object Conditional {

  def computeSize(input: Double): String = {
    val large: Double = 60.0
    val medium: Double = 30.0
    if (input >= large) {
      "large"
    } else if (input >= medium) {
      "medium"
    } else {
      "small"
    }
  }

  def main(args: Array[String]): Unit = {
    println(computeSize(70.0))
    println(computeSize(50.0))
    println(computeSize(10.0))
  }

}
```

Conditionals
Similar to JavaScript syntax

```scala
package lo1_program_execution.scala

object Conditional {

  def computeSize(input: Double): String = {
    val large: Double = 60.0
    val medium: Double = 30.0
    if (input >= large) {
      "large"
    } else if (input >= medium) {
      "medium"
    } else {
      "small"
    }
  }

  def main(args: Array[String]): Unit = {
    println(computeSize(70.0))
    println(computeSize(50.0))
    println(computeSize(10.0))
  }

}
```

Conditionals and Methods
The return value of this method is determined by the conditional

The conditional determines which expression evaluates last
-Any code after the conditional would break this example