

# Chapter 12

## Building a Class



### 12.1 Introduction

In this chapter, we will work through the creation of a simple class to represent a Company. As we are using IntelliJ, we will step through using this IDE to create our class.

### 12.2 Create a New Module

Assuming you already have a Project to work in you should add a new module to that Project. If not create a new Project using the File-> New-> Project... option from the menu bar.

Next you can create a suitable module within the IntelliJ IDE to create your application. A new module can be created from the File menu under File-> New -> Module... (Fig. 12.1).

This will cause the 'New Module' wizard to be displayed as shown below.

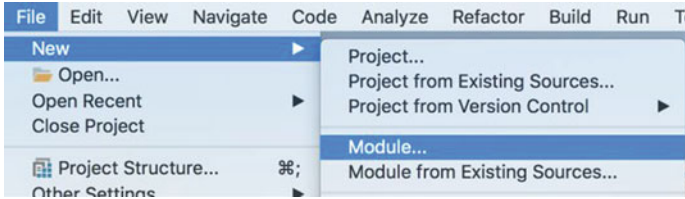
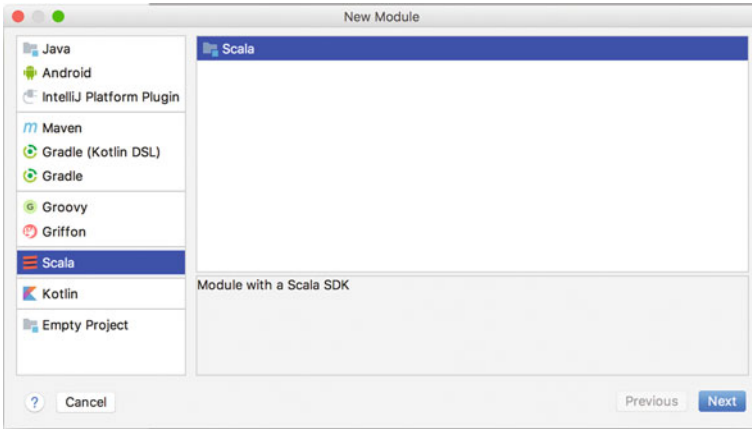


Fig. 12.1 Selecting the create a new 'Scala Project' option



Make sure that you have selected the Scala option in the left-hand window. Then click 'Next'.

You can name the module anything. In this case, we will call the Project sample. Enter the Project name into the 'Module name' field as shown here (Fig. 12.2).

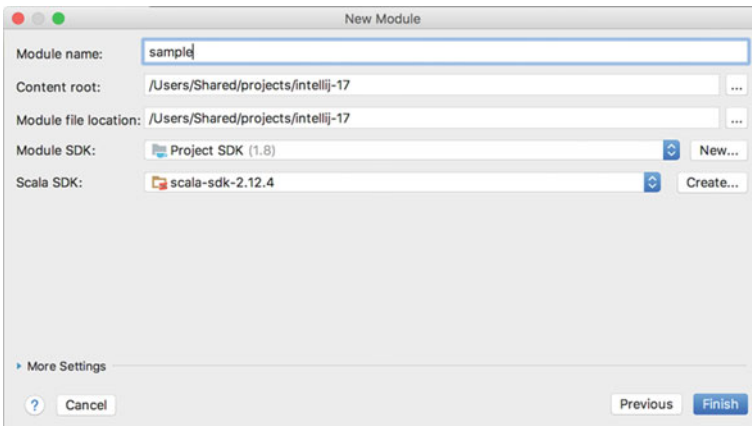


Fig. 12.2 Naming the project

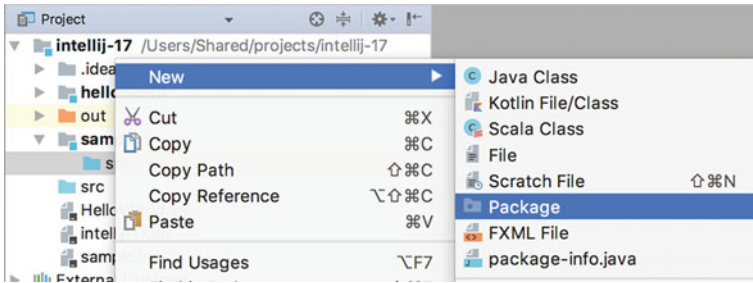


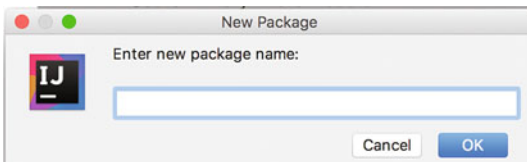
Fig. 12.3 Selecting the new package option

I would also create a package to place your code in. A package is an organisational construct that helps you manage your code. It also relates to name spacing (what can be seen where) and is a good programming technique to get into.

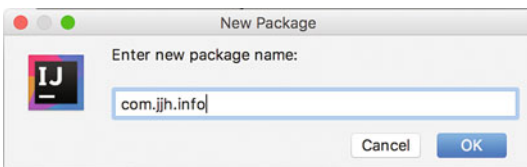
### 12.3 Create a New Package

To create a new package, select your src directory under your module and from the right mouse menu select New -> Package, for example (Fig. 12.3):

This will display the new Package Wizard (note it says Java but is being reused for Scala packages in the Scala IDE). This dialog is shown here.



You can use whatever name you wish although you should note that a Scala package is a series of names separated by '.' which are typically prefixed by the domain of the organisation creating the code. I am using com.jjh.info.



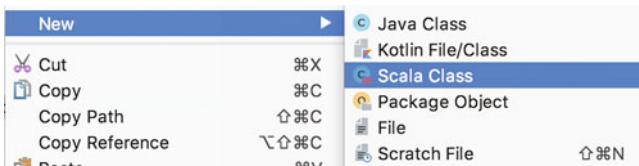
Once you have provided a package name click 'OK'.

You will now see a new package provided for you in the Project View of the IDE. An example of the structure created under the Project heading is shown below.

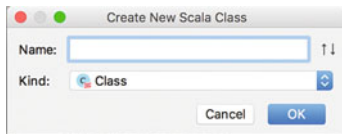


## 12.4 Create a New Class

We will now add a new class to the package we just created. This can be done using the New Scala Class Wizard. Select the package we just created in the Project View and from the right mouse menu select New -> Scala Class.



You will now be shown the 'Create New Scala Class' wizard:

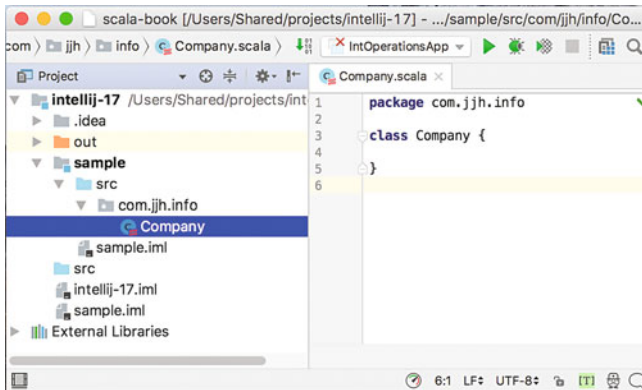


Specify the name of the class you want to create using the Name field. In our case, we will create a new Scala class called `Company` as shown below.



Now click 'OK'.

You should now find that you have a new class `Company`, in a file called `Company.scala` under the `com.jjh.info` package. While in the middle of the IDE in the code presentation area, you should see the outline skeleton code for the class `Company`.



## 12.5 Defining the Class

The simple class you just created now needs to be expanded to represent a company. The class must have the following information:

- The name of the company
- The address of the head office of the company
- The phone number of the company
- The company registration number
- The company VAT number

The address of the company could be a separate type including county, postcode/zipcode. However, we will keep things simple for the moment.

The fields of the company will all be of type `String` and will have some form of default value, for example the empty or null string represented by `""`. `String` is referred to as a type as it represents a concept with the programming language. As such a string is zero or more characters which respond to certain operations such as `substring`, `length`.

Update your definition of the `Company` class so that it resembles the following listing:

```
package com.jjh.info

class Company {
  var name = ""
  var address = ""
  var telephone = "0000"
  var registrationNumber = "000"
  var vatNumber = "xxxx"
  var postcode = "xxx xxx"
}
```

## 12.6 Adding Behaviour

We can also add some behaviour to this class by providing a `print` method that will print out the `Company` details in an appropriate format.

The printer method will be done first. This method will not return a *value* as it will be used to print information on the `Company` out to the Console.

```
package com.jjh.info

class Company {
  var name = ""
  var address = ""
  var telephone = "0000"
  var registrationNumber = "000"
  var vatNumber = "xxx"
  var postcode = "xxx xxx"

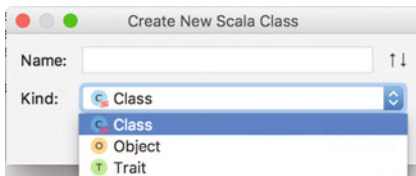
  def print() = println(s"Company name $name at $address")
}
```

Note that we have used the single line form of defining a method—this is not the only option and you could experiment with other formats one you have this version working.

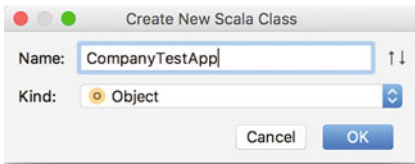
## 12.7 Test Application

You should then create a simple test application (use the `App` trait with a `Scala` object type) to create new instances of the `Company` class.

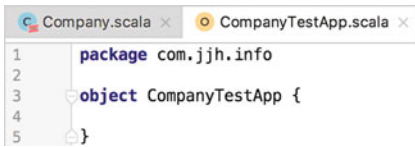
You can use the `New Scala Class` dialog to create an object as well as a class. For example, select the package again and for the right mouse menu select ‘`New Scala Class`’. However, when the dialog is displayed, select the drop-down box below the `Class Name` field.



You will then see that there are actually three options available at this point; Class, Object and Trait. Select the Object option and provide a name for the Object (I am using `CompanyTestApp`):



You should now see a new tab on in the central code editor area of the IDE as shown below.



This contains the skeleton of the `CompanyTestApp` object. It is not yet an application. Modify the declaration of the object so that it extends the `App` trait. So that you now have:

```
package com.jjh.info

object CompanyTestApp extends App {
}
```

Remember as you are using the `App` trait, you do not need to define a main method declaration—you only need to add what the application needs to do.

In our case, we will create a new instance of the `Company` class and print out its details:

```
package com.jjh.info
object CompanyTestApp extends App {
  println("Starting CompanyTestApp")
  val company = new Company()
  company.print
  println("Done CompanyTestApp")
}
```

Recall that we do not need to define the type of the `val` we will hold our `company` reference in (this will be inferred by Scala), but that new instances are created using the keyword `new`.

We can now run this application either using the right mouse menu from the file `CompanyTestApp` in the Project View (Run).

In the Run output console, you should see output similar to that shown here.



```
Run CompanyTestApp
/Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk
Starting CompanyTestApp
Company name at
Done CompanyTestApp
Process finished with exit code 0
```

This is because the `Company` object does not yet have any data defined by you. We will now add that data:

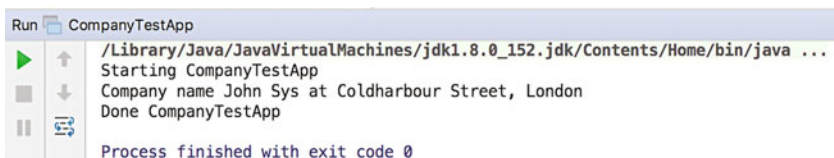
```
package com.jjh.info
```

```
object CompanyTestApp extends App {
  println("Starting CompanyTestApp")
  val company = new Company()
```

```
  // Set up the company information
  company.name = "John Sys"
  company.address = "Coldharbour Street, London"
  company.telephone = "123456"
  company.registrationNumber = "999999999"
  company.vatNumber = "BB112233AA"
  company.postcode = "BS16 1QY"
```

```
  company.print
  println("Done CompanyTestApp")
}
```

In the above example, we have populated the fields that are defined within the `Company` instance with suitable data. If you now rerun this application, you should see more comprehensible output in the Run console.



```
Run CompanyTestApp
/Library/Java/JavaVirtualMachines/jdk1.8.0_152.jdk/Contents/Home/bin/java ...
Starting CompanyTestApp
Company name John Sys at Coldharbour Street, London
Done CompanyTestApp
Process finished with exit code 0
```

## 12.8 Override ToString

In general, we would not define a custom method such as `print` to print out the `Company` instances. Typically, we would use the `println` method directly with the `company` instance. To do this, we must override the `toString` method as we did in the last chapter. In this case, our `toString` method must include



information for all of the fields. We can thus add a `toString` method to the `Company` class with the result that we can print out a `Company` instance directly. The `Company` class would now look like:

```
package com.jjh.info

class Company {
  var name = ""
  var address = ""
  var telephone = "0000"
  var registrationNumber = "000"
  var vatNumber = "xxxx"
  var postcode = "xxx xxx"

  def print() = println(s"Company name $name at $address")

  override def toString = s"Company[$name, $address, " +
    s"$telephone, $registrationNumber, $vatNumber" +
    s"$postcode]"
}
```

The test program could be updated to include a `println` for the company instance (e.g. `println(company)`):

```
package com.jjh.info

object CompanyTestApp extends App {
  println("Starting CompanyTestApp")
  val company = new Company()

  // Set up the company information
  company.name = "John Sys"
  company.address = "Coldharbour Street, London"
  company.telephone = "123456"
  company.registrationNumber = "99999999"
  company.vatNumber = "BB112233AA"
  company.postcode = "BS16 1QY"

  company.print
  println(company)
  println("Done CompanyTestApp")
}
```

The output of this program is now:

```
Starting CompanyTestApp
Company name John Sys at Coldharbour Street, London
Company[John Sys, Coldharbour Street, London, 123456,
99999999, BB112233AABS16 1QY]
123456, 99999999, BB112233AABS16 1QY]
Done CompanyTestApp
```

## 12.9 Extras

You could try out different syntax options, for example:

```
println(company name)  
company print()  
company print
```