Mobile Development with Scala

Release 1.0

Allan Davis

April 18, 2013

Contents:

CHAPTER

ONE

SETTING UP

Developing Android applications with Scala can be fun and rewarding. the code will be cleaner and simpler than the equilvant Java code. To get started, we need to install a few things. First off, the Java 6 JDK needs to be installed.

1.1 Install Java 1.6

Go to http://www.oracle.com/technetwork/java/index.html and select Java SE 6 Update 43 JDK or just go http://www.oracle.com/technetwork/java/javase/downloads/jdk6downloads-1902814.html

If you are on a Mac just type javac into a terminal to download java6 from apple.

Just follow the instructions and prompts from the java installer.

You have to use JDK 6 Android will not work if compiled with JDK 7.

1.2 Installing Android SDK

1. Go to http://developer.android.com/sdk/index.html
2. Click on "Use and existing IDE"
3. Download the SDK.
4. Extract the SDK to the root of your home directory.
5. Download the NDK
6. Extract the NDK to the root of your home directory.

1.2.1 On Mac and Linux

1. Edit .bash_profile like this.

```
export ANDROID_HOME=~/android-sdk
export ANDROID_NDK_HOME=~/android-ndk
export PATH=$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools:$ANDROID_NDK_HOME:$PATH
```

1.2.2 On Windows

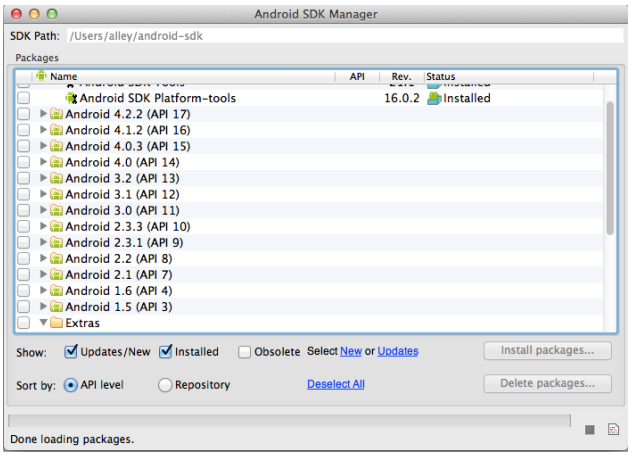
1. Right Click "My Computer"
2. Click on the "Advacned Tab"

3. Click the “Envirnment Variables” button at the bottom of the window
4. Click the “New...” button under User Variables section.
5. Set ANDROID_HOME as the variable Name and the Path to the SDK as the value
6. Click OK
7. Click The “New...” button again
8. Set ANDROID_NDK_HOME as the variable Name and the Path to the NDK as the value
9. Click OK.
10. If PATH is defined under the user block click “Edit...” else click “New...”
11. Add “%ANDROID_HOME%/tools;%ANDROID_HOME%/platform-tools;%ANDROID_NDK_HOME%;” to the Value section of path.
12. Click OK.
13. Click Apply.
14. Click OK.

1.3 Installing an Android Platform

Now we have the basic android tools installed. We need to install the platforms

```
$ android
```

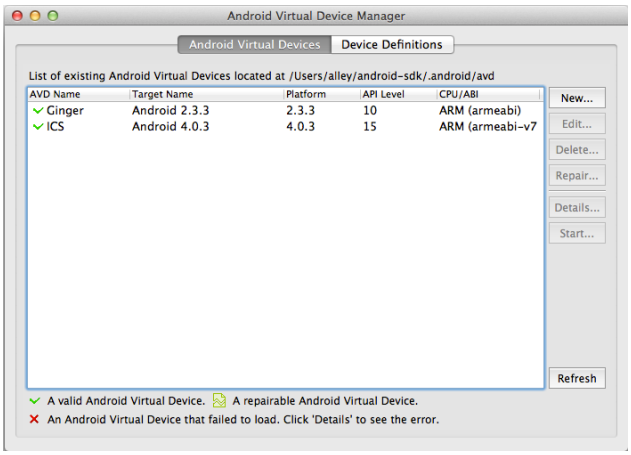


Place a chehckmark next to “Android 4.0.3 (API 15)”. This is Ice Creame Sandwitch version of android. I would select any other versions you wish to target as well. Click the Install Packages button to download the platforms. You will be asked to accept the licences.

1.4 Creating an Android Virtual Device

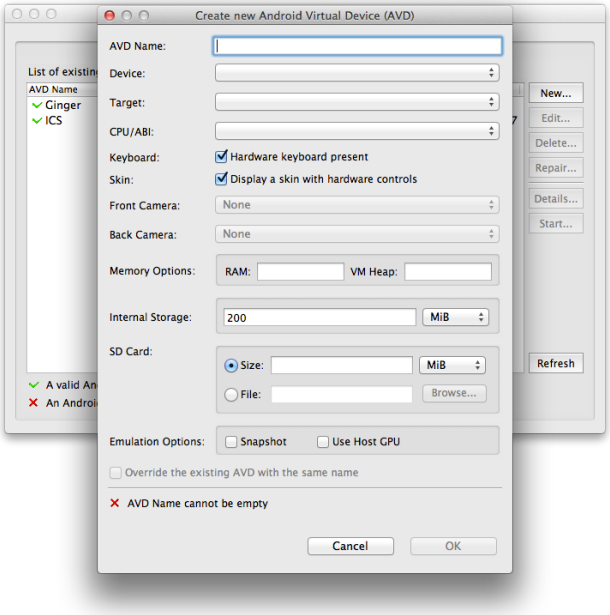
Now that we have the platforms installed, we need to create a virtual device. Android comes with a tool called the avd manager. To start the manager you type the command below.

```
$ android avd
```

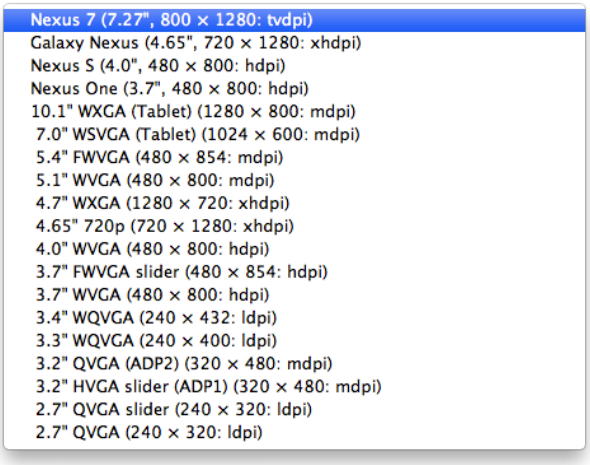


With the manager up, we can start creating the avd.

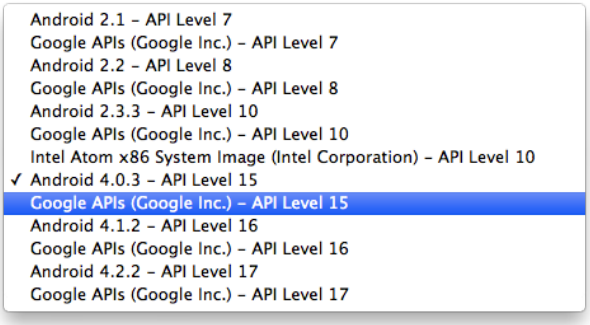
1. Click the "New..." button on the right side of the manager.



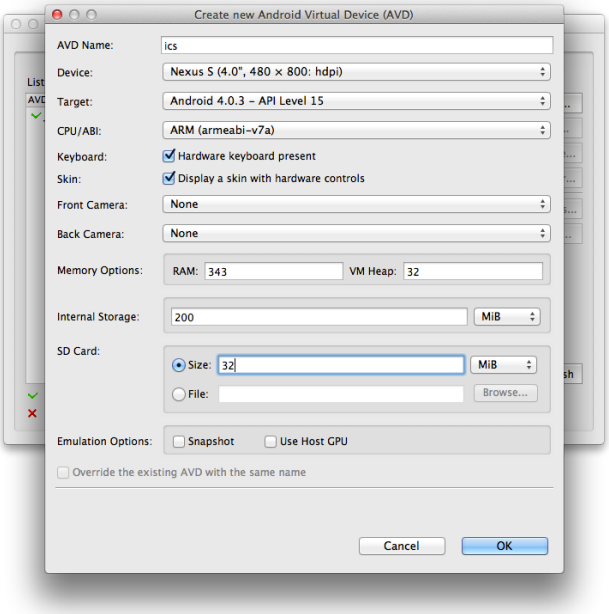
2. Give the avd a name. I ususally use the code name for the version of android that will be installed.
3. Select Nexus S from the device list.



4. Change the target name to Android 4.0.3

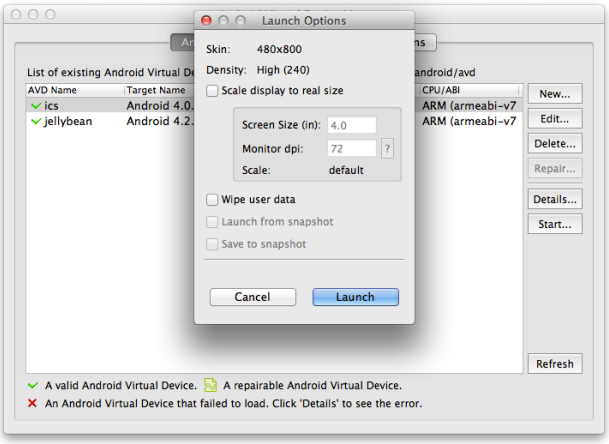


5. Set the SD Card Size to 32 Mib

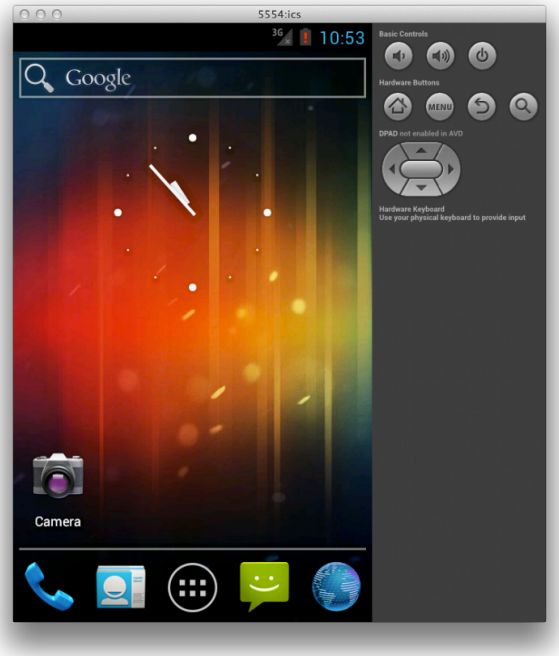


6. Click OK

7. Select the AVD you Created.
8. Click Start...



9. Click Launch



1.5 Installing Scala

1. Go to http://www.scala-lang.org/downloads
2. Download the package for your system.
3. Extract the file to the home directory

1.5.1 On Mac and Linux

1. Edit .bash_profile like this.

```
export SCALA_HOME=~/scala
export PATH=$SCALA_HOME/bin:$PATH
```

1.5.2 On Windows

1. Right Click “My Computer”
2. Click on the “Advacned Tab”
3. Click the “Envirnment Variables” button at the bottom of the window
4. Click the “New...” button under User Variables section.
5. Set SCALA_HOME as the variable Name and the Path to the Scala as the value
6. Click OK
7. If PATH is defined under the user block click “Edit...” else click “New...”
8. Add “%SCALA_HOME%/bin;” to the Value section of path.
9. Click OK.
10. Click Apply.
11. Click OK.

1.6 Installing SBT

1. Go to http://www.scala-sbt.org/release/docs/Getting-Started/Setup.html
2. Download the zip package
3. Extract the package into the home directory

1.6.1 On Mac and Linux

1. Edit .bash_profile like this.

```
export SBT_HOME=~/sbt
export PATH=$SBT_HOME/bin:$PATH
```

1.6.2 On Windows

1. Right Click “My Computer”
2. Click on the “Advacned Tab”
3. Click the “Envirnment Variables” button at the bottom of the window
4. Click the “New...” button under User Variables section.
5. Set SBT_HOME as the variable Name and the Path to the SBT as the value

6. Click OK
7. If PATH is defined under the user block click “Edit...” else click “New...”
8. Add “%SSBT_HOME%/bin;” to the Value section of path.
9. Click OK.
10. Click Apply.
11. Click OK.

1.7 Install Project Templates

1. Go to https://github.com/downloads/n8han/conscript/conscript-0.4.1.jar
2. Run this command

```
$ java -jar conscript-0.4.1.jar
```

3. If you are on Linux or Mac Run

```
$ curl https://raw.github.com/n8han/conscript/master/setup.sh | sh
```

4. Now we can install giter8 by running:

```
$ cs n8han/giter8
```

This gives us the g8 command for our templates we will use to create the projects.

1.8 Adding IDE settings

For our development we are using InteliJ IDE with the scala plugin. Also we are going to use an SBT Plugin to add ide project config generation to the tool chain.

1.8.1 Installing IntelliJ IDE

1. Go to http://www.jetbrains.com/idea/download/index.html
2. Download the Community Edition.
3. Run the installer
4. Complete the installation process

1.8.2 Installing Scala Plugin for IntelliJ

1. Open IntelliJ if not already open.
2. Open the Preferences.
3. On the Left hand side, Find and click on Plugins.
4. Click the Browse Repository button
5. In the search box type Scala

6. Right click Scala and select Download and install
7. Do the same for SBT
8. Restart the IDE

1.8.3 Configure SBT to generate IntelliJ project

Open the plugins file under the .sbt directory in the home directory. If it's not there just added it.

```
$ vim ~/.sbt/plugins/build.sbt
```

Once we have the file open we need to add this line below:

```
addSbtPlugin("com.github.mpeltonen" % "sbt-idea" % "1.3.0")
```

The next time we run sbt it will download this file and have the command available to any project.

CHAPTER

TWO

TIP CALCULATOR

Contents

2.1 Feature Scenario

```
Feature: Basic tip calculation
  Scenario: Calculate single check tip amount and total.
    Given that I want to calculate the amount of the tip
    When I enter $50 at 17% tip
    Then I should see $8.50 for the tip amount
    And $58.50 for the total amount
```

I know this looks weird. The format of this text is called Gerken. This format is used by business to describe a process they want to preform.

2.2 Project Setup

To start the project we will use the gliter8 template that we installed in the setup chapter. This command will ask a set of questions to setup the base of our project. I have my answers to the questions below. Anyting in square brackets are the default values in this template.

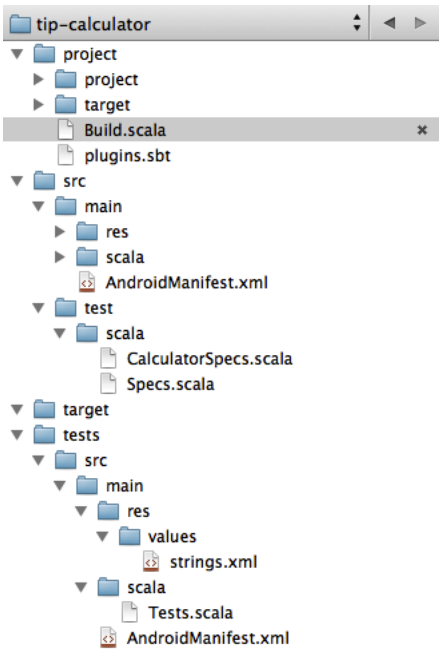
```
$ ~/bin/g8 jberkel/android-app
Template for Android apps in Scala

package [my.android.project]: com.cajuncode.tipcalculator
name [My Android Project]: TipCalculator
main_activity [MainActivity]:
scala_version [2.9.2]: 2.10.0
api_level [10]:
useProguard [true]:
scalatest_version [1.8]: 1.9.1

Template applied in ./tip-calculator
```

The first thing we give the project is the package bundle identifier of the project. You can change this at any time untill you submit the app to the play store. After that if you chnage this id it will be a new app and not an update. Next is the name of the app. Use Camel case for naming the project and all Class files. Camel case is a naming convention where each word in a name is capitalized and there is no space between words. Next we need to name the starting screen for the app. Personally I would use HomeActivity or MainActivity depending on how many Activities the app will have. We will discusses activities in the next section as we start laying out the interface. After we set the activity we need to tell it what version of Scala we want to ues. Then we select the Android API level we want to use. This basicly is what version of android you are targeting with the application. We want to use Proguard, it makes our apps smaller by removing unsused classes. Last is the version of Scalatest we are going to use to write our tests.

This creates a working project for us.



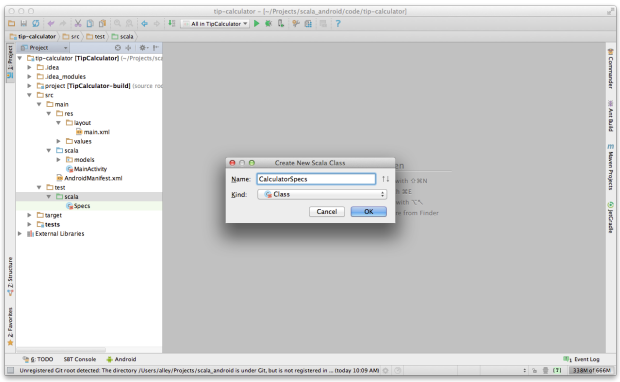
Lets create the IDE project settings.

```
$ sbt gen-idea
```

Next lauhch IntelliJ and click Open Project. Navigate to the directory we created the tipCalcualtor app. Click on the directory name, then click open. The IDE should open with the project enabled.

2.3 Creating the CalculatorCore

Now we need to create a test file. In the project window, expand the src directory. Next expand the test directory and right click on the scala diretory. Select the Scala Class option.



Now we need to add the folling lines to the scala test file we just created.

```
import org.scalatest.FunSpec
import org.scalatest.matchers.ShouldMatchers

class CalculatorSpecs extends FunSpec with ShouldMatchers {
  describe("a Tip Calculator") {
  }
}
```

Before we go and write the first test. we need to create the basic object we are testing.

1. In the project window, expand the src directory.
2. Then expand the main directory.
3. Right click on the scala directory.
4. Select New > Package...
5. Give the pckage the name models
6. Click OK
7. Right click on the models package we just created.
8. select New > Scala Class...
9. Name the class CalculatorCore
10. Set the type to be Object.
11. Click OK

```
  package com.cajuncode.tipcalculator.models

  object CalculatorCore {

  }
```

Now that we have an object to test, lets create our first test. Open the CalculatorSpecs file and add the following lines inside the describe block.

```
it("should calculate the tip of 100.00"){
  val calc = CalculatorCore
  var result = calc.getTip(100.00, 10)
  result should equal(10.00)
}
```

Ok lets run the test

```
$sbt test
[info] Compiling 2 Scala sources to /Users/alley/Projects/scala_android_book/code/tip-calculator/targ
[error] /Users/alley/Projects/scala_android_book/code/tip-calculator/src/test/scala/CalculatorSpecs.s
[error]       var result = calc.getTip(100.00, 10)
[error]                            ^
[error] one error found
[error] (TipCalculator/test:compile) Compilation failed
[error] Total time: 21 s, completed Apr 17, 2013 12:31:53 AM
```

This tells us that the method needs to be defined. So that is where we are going to start.

```
def getTip(bill:Double, tip_percentage:Int):Double = {
   0.00
}
```

run the test again.

```
$ sbt test
[info] Loading global plugins from /Users/alley/.sbt/plugins
[info] Loading project definition from /Users/alley/Projects/scala_android_book/code/tip-calculator/p
[info] Set current project to TipCalculator (in build file:/Users/alley/Projects/scala_android_book/c
[info] Wrote /Users/alley/Projects/scala_android_book/code/tip-calculator/target/scala-2.10/src_manag
[info] Compiling 1 Scala source to /Users/alley/Projects/scala_android_book/code/tip-calculator/targe
[info] Compiling 2 Scala sources to /Users/alley/Projects/scala_android_book/code/tip-calculator/targ
[info] Specs:
[info] a spec
[info] - should do something
[info] CalculatorSpecs:
[info] a Tip Calculator
[info] - should calculate the tip of 100.00 *** FAILED ***
[info]   0.0 did not equal 10.0 (CalculatorSpecs.scala:10)
[error] Failed: : Total 2, Failed 1, Errors 0, Passed 1, Skipped 0
[error] Failed tests:
[error]      CalculatorSpecs
```

Lets refactor the method to return a live value. Change the getTip method to look like below.

```
def getTip(bill:Double, tip_percentage:Int): Double = (bill * (tip_percentage / 100.0))
```

Now lets run the test again.

```
$ sbt test
[info] Loading global plugins from /Users/alley/.sbt/plugins
[info] Loading project definition from /Users/alley/Projects/scala_android_book/code/tip-calculator/p
[info] Set current project to TipCalculator (in build file:/Users/alley/Projects/scala_android_book/c
[info] Wrote /Users/alley/Projects/scala_android_book/code/tip-calculator/target/scala-2.10/src_manag
[info] Compiling 1 Scala source to /Users/alley/Projects/scala_android_book/code/tip-calculator/targe
[info] Specs:
[info] a spec
[info] - should do something
[info] CalculatorSpecs:
[info] a Tip Calculator
[info] - should calculate the tip of 100.00
[info] Passed: : Total 2, Failed 0, Errors 0, Passed 2, Skipped 0
[success] Total time: 4 s, completed Apr 17, 2013 1:00:00 AM
```

And we have passing our first passing test. So lets create one more test.

In the feature story we have at the start of the chapter, We use a 50 dollar balance with a 17% tip so lets write that as a test.

Open the CalculatorSpecs file and add this below the closing bracket after the it.

```
it("should calculate the tip of 50 dollars at 17% to equal 8.5"){
  val calc = CalculatorCore
  var result = calc.getTip(50.00, 17)
  result should equal(8.5)
}
```

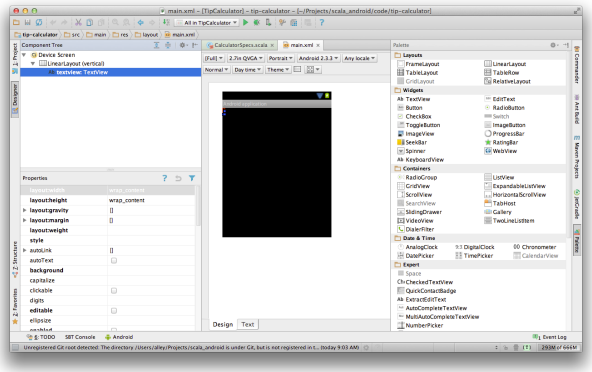
Now run the test.

```
$ sbt test
[info] Loading global plugins from /Users/alley/.sbt/plugins
[info] Loading project definition from /Users/alley/Projects/scala_android/code/tip-calculator/projec
[info] Set current project to TipCalculator (in build file:/Users/alley/Projects/scala_android/code/t
[info] Wrote /Users/alley/Projects/scala_android/code/tip-calculator/target/scala-2.10/src_managed/ma
[info] Compiling 3 Scala sources and 1 Java source to /Users/alley/Projects/scala_android/code/tip-ca
[warn] there were 4 feature warnings; re-run with -feature for details
[warn] one warning found
[info] Compiling 2 Scala sources to /Users/alley/Projects/scala_android/code/tip-calculator/target/sc
[info] Specs:
[info] a spec
[info] - should do something
[info] CalculatorSpecs:
[info] a Tip Calculator
[info] - should calculate the tip of 100.00
[info] - should calculate the tip of 50 dollars at 17% to equal 8.5
[info] Passed: : Total 3, Failed 0, Errors 0, Passed 3, Skipped 0
[success] Total time: 20 s, completed Apr 17, 2013 9:03:06 AM
```

Now that we know the calculation works, Time to work on the UI.

2.4 Designing the User Interface

Open the main.xml file under src/main/res/layout. When the file first opens in design mode. you see a palette on the right side of the screen. On the left side, The component tree shows all the components in the layout. Also on the left side is the property window. This is where you will change properties such as text and values for the components.



Here is the xml use in the view. it would be a lot of steps if we built the UI though the designer.

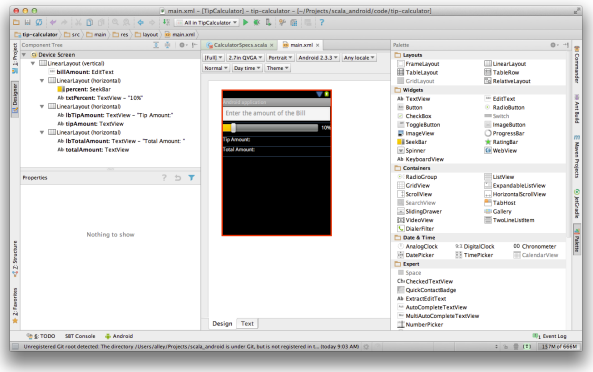
```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content" android:orientation="vertical">

    <EditText
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:id="@+id/billAmount" android:inputType="numberDecimal" android:hint="Enter the ar
            android:gravity="right"/>
    <LinearLayout
            android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent" android:paddingBottom="10dp">
        <SeekBar
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:id="@+id/percent" android:layout_weight="3" android:paddingRight="10dp" andro
                android:progress="17"/>
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="17%"
                android:id="@+id/txtPercent" android:layout_gravity="center"/>
    </LinearLayout>
    <LinearLayout
            android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent" android:paddingBottom="10dp">
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Tip Amount:"
                android:id="@+id/lbTipAmount" android:layout_weight="2" android:paddingLeft="3dp"
                android:password="false"/>
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:id="@+id/tipAmount" android:layout_weight="2" android:gravity="right"
                android:paddingRight="10dp"/>
    </LinearLayout>
    <LinearLayout
            android:orientation="horizontal"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent" android:paddingBottom="10dp">
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Total Amount: "
                android:id="@+id/lbTotalAmount" android:layout_weight="2" android:paddingLeft="3dp"/>
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:id="@+id/totalAmount" android:layout_weight="2" android:gravity="right"
                android:paddingRight="10dp"/>
    </LinearLayout>
</LinearLayout>
```

With the UI Layed out this it what it looks like in the designer.



OK we are getting close enough to try and running the app and see it. But first we need to fix a piece of code so the app will launch. Under src/main/scala/MainActivity.scala, we need to comment out the findView call on line 11 like so.

```
1  package com.cajuncode.tipcalculator
2
3  import _root_.android.app.Activity
4  import _root_.android.os.Bundle
5
6  class MainActivity extends Activity with TypedActivity {
7    override def onCreate(bundle: Bundle) {
8      super.onCreate(bundle)
9      setContentView(R.layout.main)
10
11     //findView(TR.textview).setText("hello, world!")
12   }
13 }
```

Before we can run the app we need to start the emulator

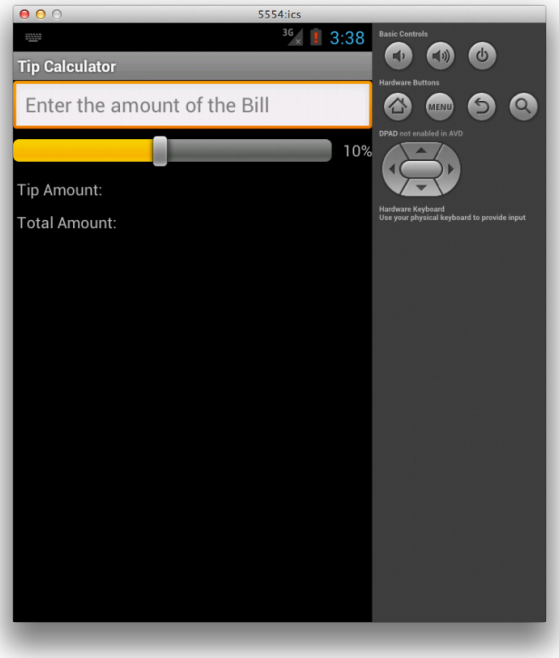
```
$ emulator -avd ics
```

This will start the emulator and load the ics avd.

With the emulator started we want to push our application to the emulator. On a different terminal window or command prompt run the following command.

```
$ sbt android:start-emulator
[info] Loading global plugins from /Users/alley/.sbt/plugins
[info] Loading project definition from /Users/alley/Projects/scala_android/code/tip-calculator/projec
[info] Set current project to TipCalculator (in build file:/Users/alley/Projects/scala_android/code/t
[info] Wrote /Users/alley/Projects/scala_android/code/tip-calculator/target/scala-2.10/src_managed/ma
ProGuard, version 4.6
ProGuard is released under the GNU General Public License. You therefore
must ensure that programs that link to it (scala, ...)
carry the GNU General Public License as well. Alternatively, you can
apply for an exception with the author of ProGuard.
The output seems up to date
    (skipping file '.gitkeep' due to ANDROID_AAPT_IGNORE pattern '.*')
[info] Packaging /Users/alley/Projects/scala_android/code/tip-calculator/target/tipcalculator-0.1.apk
[info]       pkg: /data/local/tmp/tipcalculator-0.1.apk
[info] Success
[info] 1847 KB/s (181817 bytes in 0.096s)
[info] Starting: Intent { act=android.intent.action.MAIN cmp=com.cajuncode.tipcalculator/.MainActivit
[success] Total time: 17 s, completed Apr 17, 2013 3:26:29 PM
```

This packages the application into an apk and deploys it to the emulator. Rerunning this command will redeploy the app. It is a good practice not to restart the emulator each deployment due to is painful boot time.



Now we see what we've built, It's time to bind the components to the views. Go back to the MainActivity

```
...
import android.widget.{TextView, SeekBar, EditText}

class MainActivity extends Activity with TypedActivity {

  private lazy val bill:EditText = findView(TR.billAmount)
  private lazy val percentage:SeekBar = findView(TR.percent)
  private lazy val percentText:TextView = findView(TR.txtPercent)
  private lazy val tipAmount:TextView = findView(TR.tipAmount)
  private lazy val totalAmount:TextView = findView(TR.totalAmount)

  ...
```

Lets add variables to point to the components we want to bind. findView is a TypedActivity method to add a helper to do object casting.

With the variables bound we can turn our attention to onCreate. This method is run when the Activity is first started. An Activity is what android calls a window that is to be displayed.

```
override def onCreate(bundle: Bundle) {
  super.onCreate(bundle)
  setContentView(R.layout.main)

  bill.addTextChangedListener( new TextWatcher {
    def beforeTextChanged(value: CharSequence, start: Int, count: Int, after: Int) {}

    def onTextChanged(value: CharSequence, start: Int, before: Int, count: Int) {}

    def afterTextChanged(value: Editable) {
      updateTip()
```

```
    }
  } )
  percentage.setOnSeekBarChangeListener(new OnSeekBarChangeListener {
    def onProgressChanged(seekbar: SeekBar, progress: Int, fromUser: Boolean) {
      updateTip()
      percentText.setText(progress.toString + "%")
    }

    def onStopTrackingTouch(seekbar: SeekBar) {}

    def onStartTrackingTouch(seekbar: SeekBar) {}
  })
}
```

In this method we use inner classes to implement two listeners. Each Listener is a contract that states that a set of methods will be created. This is offten called an Interface. The only real events we care about listening for is the after text has changed in the billAmount edit box and when the seek bar has changed. For both of these events we want to call updateTip method.

```
def updateTip(){
  val billText:String = bill.getText().toString
  val amount = convert(billText)

  val tip = CalculatorCore.getTip(amount, percentage.getProgress)
  tipAmount.setText(formatDecimal(tip))
  totalAmount.setText(formatDecimal(amount + tip))
}
def convert(value:String):Double = if(value.isEmpty) 00 else value.toDouble
def formatDecimal(value:Double):String = "%1.2f" format value
```

With updateTip defined all we do is read in the value from the edit box and convert it to a string. Next we will convert the text to a double precision floating point number or double. This conversion has been wrapped into a function to handle the case of updating the field to empty. Then using the model we created, calculated the tip amount. Now all that is left is to display the tip and the total back to the user. The formatDecimal just formats the amounts calculated to a currency format with two decimal places.

Warning: java.lang.NoSuchMethodError: scala.collection.immutable.StringLike.toString
As I was working on this. I stumbled upon a bug in Scala with Proguard. When proguard runs trimming the code it removes a method necessary for the convertion to Double from string to work. To fix this you need to make a change to the proguard config in the Build.scala file under project. Under object General add a new val proguardOptions with the keep class string. Then add that to the fullAndroidSettings as seen below. This is a known bug in Scala 2.10.0

```
val proguardOptions = Seq(
  proguardOption in Android := "-keep class scala.collection.SeqLike {\n    public protected *;\n}"
 )
lazy val fullAndroidSettings =
  General.settings ++
  AndroidProject.androidSettings ++
  TypedResources.settings ++
  proguardSettings ++
  proguardOptions ++
  AndroidManifestGenerator.settings ++
  AndroidMarketPublish.settings ++ Seq (
    keyalias in Android := "change-me",
    libraryDependencies += "org.scalatest" %% "scalatest" % "1.9.1" % "test"
  )
```

Now lets run the app again. with the emulator running type the following command.

```
$ sbt android:start-emulator
```