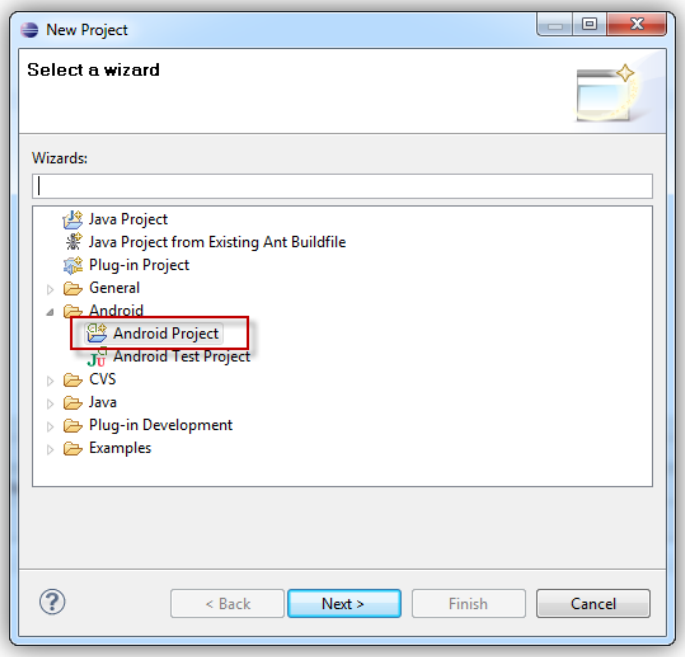
Working with Classes

In this lesson, we're going to look at Java classes. You'll learn how to create a class, how to use it within your application, and how classes may inherit from one another. We'll talk about **Mutators**, **Accessors**, and **Constructor** methods as well.

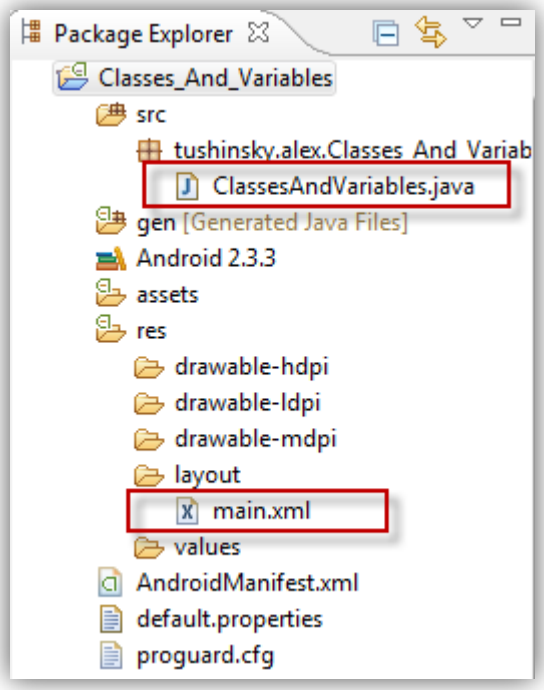
Begin by creating a new project in Eclipse by selecting the **File** menu, then **New** option, then **Project**. Highlight **Android Project** in the New Project wizard, and click **Next**.



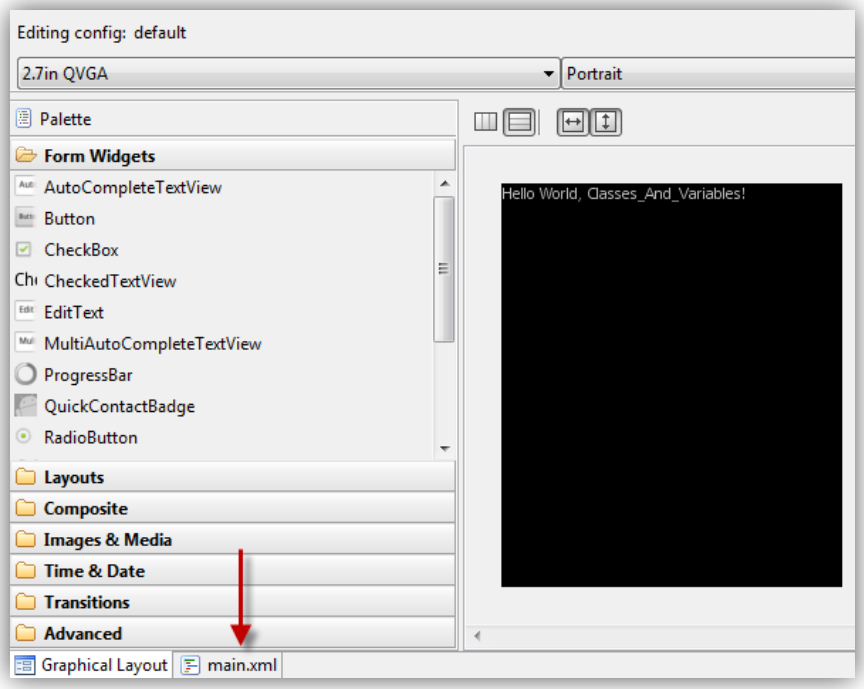
Provide the following information:

Project name: **Classes_and_Variables**
Create new project in workspace should be selected.
Built Target: **Android 2.3.3**
Application name: **Classes_and_Variables**
Package name: **lastname.firstname.Classes_and_Varaibles** (substitute lastname.firstname with your last name and first name – no spaces, apostrophes or dashes.)
Create Activity should be checked, and **ClassesAndVariables** should be the prompt's text.
Min SDK Version: **10**

Expand the project's tree (in Project Explorer). We'll be working in the **src /package/** and the **res/layout/** folders today. Start by opening the main.xml file found in res/layout/ folder.



Switch to the "**main.xml**" tab, found at the bottom of the layout view.



Mody the <TextView> element's XML to look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/txtOut"
    />
</LinearLayout>
```

A LinearLayout is one where all of the elements on the screen are positioned either vertically or horizontally. In our case, the view is veritical (note the orientation property). So any element found inside the LinearLayout element will be positioned vertically in the same order as it was setup in the XML file. In our case, we only have one element, which is a TextView.

A TextView is responsible for outputting text on the screen, much like a label. It does not allow you to enter text; it is for display purposes only. In order for us to control its output, we need to be able to reference this TextView using code. By specifying the id property, we've done exactly that. Now, we can find and use the "txtOut" TextView in our code, and whatever we tell it to display, will appear on the phone's screen.

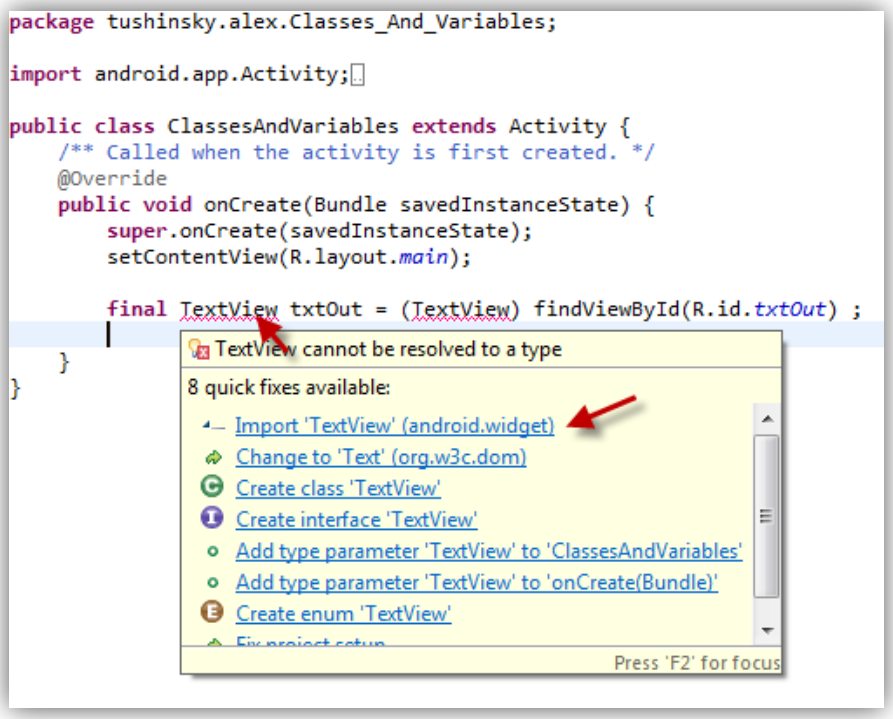
Save and close the main.xml file. We won't need to make any further changes to it today.

Next, open the **ClassesAndVariables.java** file, found in the **src/package name** folder of your project.

Add the following line to the onCreate event.

```
package tushinsky.alex.Classes_And_Variables;

import android.app.Activity;
import android.os.Bundle;

public class ClassesAndVariables extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final TextView txtOut = (TextView) findViewById(R.id.txtOut);

    }
}
```

Now, look at the word “TextView” that appears right after “final”. Notice that it’s underlined in red. In Eclipse, this means that we have an error in our code. In this particular case, the TextView class necessary to create an instance of a TextView variable is not imported into our project. The easiest way to fix this, is to hover over the word “TextView”. Eclipse will provide us with a quick menu of choices to select from that will resolve the error. The first choice is to add an import statement “**Import ‘TextView (android.widget)**”. Select this link and the error will be resolved.



What we’ve done with this line is created a pointer to our XML’s TextView control using a built-in method called FindViewById. We stored the pointer reference for this object in a local variable that we created, also called txtOut.

Here’s a breakdown of the line:

final - Means that this is a constant that isn’t going to change.
TextView – The type of variable we’re creating. In this case it’s going to be a TextView.
txtOut – The name of our variable.
= (TextView) – findViewByID returns a reference to an object in XML. It doesn’t really know or care about the type of object it’s returning. So we cast the returned object from findViewById as a TextView.
findViewById(R.id.txtOut); - This is the function that goes off and locates the object within our XML file(s). Without an ID, it would never be able to find it, which is why our first step in this lesson was to go and name the TextView in our XML file by providing it with an “android:id” property. Notice that the

R.id.txtOut appeared for you in a drop-down. This means that Eclipse, and your project is aware of this object. This was compiled for you automatically into the "R" class that is maintained for you by Eclipse. R stands for resource. It is a complete index of resources that are defined in your application. You can view this class by expanding the **gen/package/** folder and opening the R.java class file. Do not make any changes to this file.

Building A Class

Our next mission is going to be to build a class. We're going to define a simple class that shows us some basic properties of a computer. We're interested in looking at the following properties:

- CPU Speed in Mhz.
- Hard Drive size in GB.
- Screen Resolution (width x height) in pixels
- RAM size in GB
- PC Name

Once the above attributes are collected, we'd like to output them to screen in the following format:

```
Gateway ABC123
RAM: 4GB
Speed: 1.2 Ghz Processor
HD Size: 500 GB
Screen Res: 1920 x 1080
```

To build a class that accomplishes this, perform the following steps:

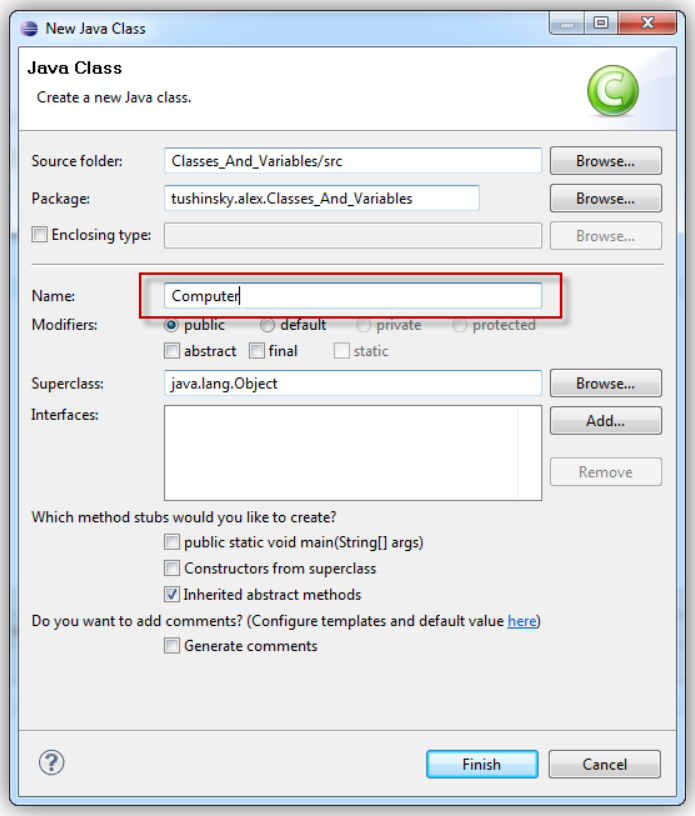
Select File → New → Class.

Provide the following information:

Source folder: **Classes_And_Variables/src** (should already be set)
Package: **Should be your package name** (should already be set)
Name: **Computer**

Click **Finish**.

A new file will be created for you with the filename / class being called Computer.



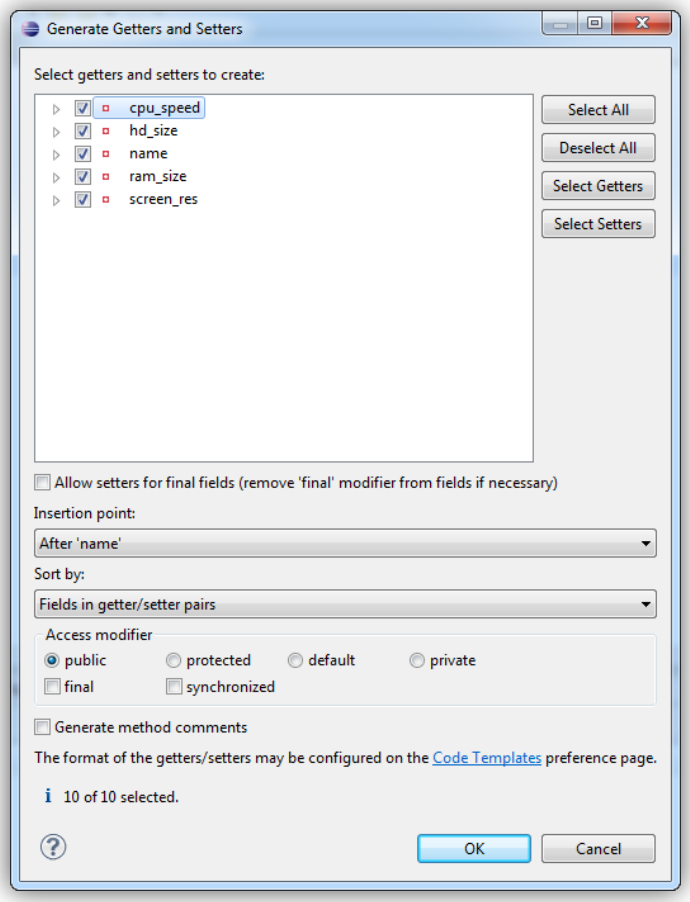
Enter the following code to make your file look like this:

```
package tushinsky.alex.Classes_And_Variables;

public class Computer {

	private int cpu_speed;
	private int hd_size;
	private String screen_res;
	private int ram_size;
	private String name;

}
```

What we've done is create the internal class variables that will be used to store our data values. These values are not exposed outside of our class, because they are labeled private.

Right-click on an empty spot in the code window, and select **Source**, then **Generate getters and setters**.

This is a shortcut for us. Instead of typing in all of the code necessary for us to manage values within our class, we'll let Eclipse do the work for us. Select all options, and click **OK**.



Your code should look like this:

```
package tushinsky.alex.Classes_And_Variables;

public class Computer {

	private int cpu_speed;
	private int hd_size;
	private String screen_res;
	private int ram_size;
	private String name;

	public int getCpu_speed() {
		return cpu_speed;
	}
	public void setCpu_speed(int cpu_speed) {
		this.cpu_speed = cpu_speed;
	}
	public int getHd_size() {
		return hd_size;
	}
	public void setHd_size(int hd_size) {
		this.hd_size = hd_size;
	}
	public String getScreen_res() {
		return screen_res;
	}
	public void setScreen_res(String screen_res) {
		this.screen_res = screen_res;
	}
	public int getRam_size() {
		return ram_size;
	}
	public void setRam_size(int ram_size) {
		this.ram_size = ram_size;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}

}
```

The methods that start with “get” are the “getters” or as they are officially known as “Accessors”. These methods allow us to retrieve values from our class. Note that each returns a value.

The methods that start with “set” are the “setters” or “Mutators”. These methods allow us to set or change the values of our classes’ private variables. In our case, these are the five variables we defined originally. These methods are all known as constructor methods.

Also, note the fact that we're using the same names in our set methods. For example, look at the line that reads this.ram_size = ram_size;

this.ram_size refers to the private variable using within the class, while ram_size after the equals refers to the local variable being passed in as a parameter to the setRam_size method. This is part of variable scope we talked about in the presentation. Local ram_size is only known while the setRam_size function is running. Once it's done, only this.ram_size is known, because that variable is global to the whole class.

Next, let's modify some of these methods, and provide a toString() output function for our Computer class.

Modify the getCpu_speed routine. We're going to change the data type from int to double. This is necessary for us in order to properly convert Mhz to Ghz. Consider 1200Mhz, which would be 1.2Ghz. Modify the return value by dividing it by 1000. We need to cast cpu_speed as a double in order to preserve the value behind the decimal point.

```
public double getCpu_speed() {
	return (double)cpu_speed / 1000;
}
```

Now, let's add the toString() method that we'll use for output. Typically, all classes should have a toString method as a default.

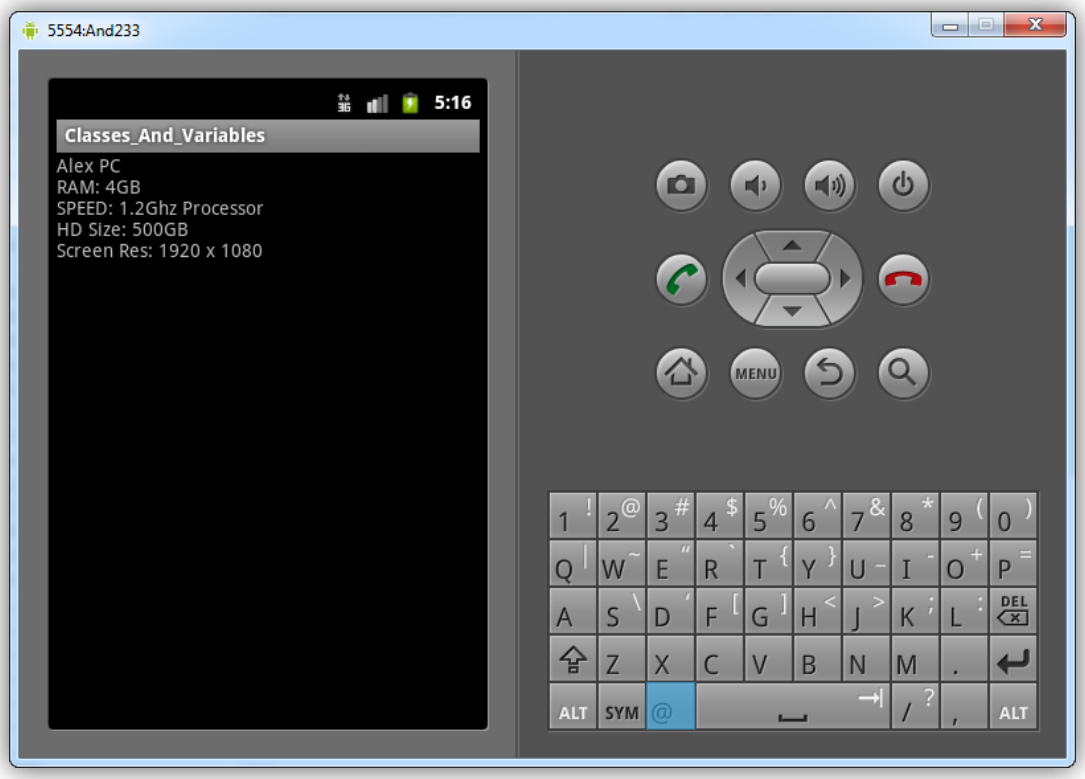
Towards the bottom of your class, add the following code:

```
	public void setName(String name) {
		this.name = name;
	}

	public String toString() {
		String sText = "";

		sText += getName() + "\n" +
			"RAM: " + getRam_size() + "GB\n" +
			"SPEED: " + getCpu_speed() + "Ghz Processor\n" +
			"HD Size: " + getHd_size() + "GB\n" +
			"Screen Res: " + getScreen_res() + "\n";

		return sText;
	}

}
```

Now, let's test out our class. Go back to ClassesAndVariables.java file, and add the following code:

```
package tushinsky.alex.Classes_And_Variables;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class ClassesAndVariables extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        final TextView txtOut = (TextView) findViewById(R.id.txtOut) ;

        Computer oPC = new Computer();
        oPC.setCpu_speed(1200);
        oPC.setHd_size(500);
        oPC.setName("Your Name PC");
        oPC.setRam_size(4);
        oPC.setScreen_res("1920 x 1080");
        txtOut.setText(oPC.toString());
    }
}
```

Here's what we've done. First, we created an instance of the Computer class, then used the Mutator methods to set values for each of the properties (or states as they are officially known). The last line outputs the toString() method of our classes's instance out to the Android screen. The results should look like this:



We can create as many different Computer objects in our code as we need at this point. Each one will be an instance of the Computer class, and each one can have different values.

Our next step is going to be to create a Notebook class. The notebook class will extend from the Computer class, because they share the same properties. The only thing we'll add is a new property for form factor, which will determine if the notebook is a Notebook, sub-notebook, or a tablet pc.

Create a new class using the same method as when we created the computer class. Call it "**Notebook**".

Modify the notebook class to look like this:

```
package tushinsky.alex.Classes_And_Variables;

public class Notebook extends Computer {

	private String form_factor;

	public String getForm_factor() {
		return form_factor;
	}

	public void setForm_factor(String form_factor) {
		this.form_factor = form_factor;
	}

	public String toString() {
		String sText = "";

		sText += getName() + "\n" +
				"RAM: " + getRam_size() + "GB\n" +
				"SPEED: " + getCpu_speed() + "Ghz Processor\n" +
				"HD Size: " + getHd_size() + "GB\n" +
				"Form Factor: " + getForm_factor() + "\n" +
				"Screen Res: " + getScreen_res() + "\n";

		return sText;
	}

}
```

Note the “extends Computer” addition to the class declaration. Here, we told the Notebook class to inherit the properties of the Computer class. This allows our toString() method to work, even though the Notebook class does not have the definitions for getRam_size(), getCpu_speed(), etc.

Save your work, and return to ClassesAndVariables.java. Add the following code to test out your new class:

```
        oPC.setHd_size(500);
        oPC.setName("Your Name PC");
        oPC.setRam_size(4);
        oPC.setScreen_res("1920 x 1080");

        Notebook oNote = new Notebook();
        oNote.setCpu_speed(1100);
        oNote.setHd_size(320);
        oNote.setName("My Notebook");
        oNote.setRam_size(2);
        oNote.setScreen_res("1280 x 720");
        oNote.setForm_factor("Tablet PC");

        txtOut.setText(oPC.toString() + "\n" oNote.toString());
```

Your output will look something like this:

Be sure to review the video accompanying this lesson for additional information.