Skills and Topics for TeenCoder: Android Programming

Our Self-Study Approach

Our courses are **self-study** and can be completed on the student's own computer, at their own pace. You can steer your student in the right direction with no prior programming knowledge. Students only need typical computer usage skills to start; we will teach them programming from the ground up!

Each course comes with student activity starters, supplementary instructional documents, a **Solution Guide**, fully coded solutions for all activities, tests and answer keys, and guidance on evaluating projects.

Most questions about how to code individual activities are easily answered by referring to the Solution Guide (with or without parental involvement). We also provide **free technical support** to assist with any aspect of the courses!

Teachers who wish to closely monitor and grade student progress for credit purposes can administer chapter tests which are provided (with answer keys). We also provide advice and guidelines for evaluating student activities.

What Skills do Students Need to Begin?

All of our courses assume the student is already familiar with using a keyboard and mouse to select and run software, navigate the menus in a typical software program, and generally interact with their computer.

Students should understand how to use the built-in operating system software (Windows Explorer or Mac Finder) to find, save and retrieve files on their computer. It may also be helpful to have some familiarity with text editors (like Notepad or TextEdit) and some experience using web browsers to find information on the Internet. We teach students how to program a computer from the ground up, but they should already know the basics about using one!

This course requires a **Windows** or **Mac OS** computer with CD-ROM or DVD-ROM.

TeenCoder: Android Programming is a second-semester course. **Students must have successfully completed the first-semester TeenCoder: Java Programming course prior to starting TeenCoder: Android Programming!**

Topics Covered In This Course

The following are some of the computer programming topics that are covered in this course. For a full list of topics and sections, please see the Table of Contents for this course.

- Introduction to the Android Development Tools
- Understanding and editing XML files
- Creating and switching between screens
- Managing Android screen layouts
- Using graphical Android UI widgets
- Saving data to internal storage and SD cards
- Debugging using the emulator
- Displaying images and loading image resources
- Progress, date/time, and alert dialogs
- Menus and notifications
- SMS messaging and networking concepts
- Creating home app widets
- Publishing to the Android Market

TeenCoder™ Series

Android™ Programming

An introduction to writing
Android programs
for high school students.

Student Textbook

Second Edition

Compatible with
Mac and Windows

TeenCoder™ Series

TeenCoder™: Android Programming

Student Textbook

Second Edition

Homeschool Programming, Inc.

TeenCoder™: Android Programming

Second Edition

980 Birmingham Rd, Suite 501-128
Alpharetta, GA 30004

ISBN: **978-0-9830749-8-4**

"Android" is a trademark of Google, Inc.

Terms of Use

Disclaimer

Homeschool Programming, Inc, and their officers and shareholders, assume no liability for damage to personal computers or loss of data residing on personal computers arising due to the use or misuse of this course material. Always follow instructions provided by the manufacturer of 3rd party programs that may be included or referenced by this course.

Contact Us

You may contact Homeschool Programming, Inc. through the information and links provided on our website: http://www.HomeschoolProgramming.com. We welcome your comments and questions regarding this course or other related programming courses you would like to study!

Other Courses

Homeschool Programming, Inc. currently has two product lines for students: KidCoder™ and TeenCoder™. Our KidCoder™ Series provides easy, step-by-step programming curriculum for 4th through 12th graders. The Visual Basic series teaches introductory programming concepts in a fun, graphical manner. The Web Design series lets students create their own websites in HTML. Our TeenCoder™ Series provides introductory programming curriculum for high-school students. These courses are college-preparatory material designed for the student who may wish to pursue a career in Computer Science or enhance their transcript with a technical elective. Students can learn C#, Java, game programming, and Android application development.

3rd Party Copyrights

Instructional Videos

This course may be accompanied by optional Instructional Videos. These Flash-based videos will play directly from a DVD drive on the student's computer. Instructional Videos are supplements to the Student Textbook, covering every chapter and lesson with fun, animated re-enforcement of the main topics.

Instructional Videos are intended for students who enjoy a more audio-visual style of learning. They are not replacements for the Student Textbook, which is still required to complete this course! However by watching the Instructional Videos first, students may begin each textbook chapter and lesson already having some grasp of the material to be read. Where applicable, the videos will also show "screencasts" of a real programmer demonstrating some concept or activity within the software development environment.

This Student Textbook and accompanying material are entirely sufficient to complete the course successfully! Instructional Videos are optional for students who would benefit from the alternate presentation of the material. For more information or to purchase the videos separately, please refer to the product descriptions on our website: http://www.HomeschoolProgramming.com.

Living on the Edge!

The Android OS and Android devices are relatively new, rapidly changing technologies. That means the programming tools may not be as stable or mature as older technologies. Part of the fun and excitement when dealing with cutting-edge innovations is keeping up with the changes and overcoming anything unexpected in the development environment. You might find that certain features don't work exactly the way the Android documentation says they should, or that the software emulator is a bit quirky. We'll guide you through the obstacles we have found and overcome, but you might encounter your own challenges!

Newer versions of the Android Software Development Kit (SDK) and related packages are released over time. **Our course uses a specific version of each component and for best results we highly recommend you use exactly the same component versions.** Detailed installation instructions are available on our website (www.HomeschoolProgramming.com); please follow them carefully to ensure a smooth course experience. If you find any components or 3rd party APIs do not behave as documented, please contact us for updated instructions.

Students are encouraged to explore the online Android reference material and other information at http://developer.android.com/index.html. The tabs across the top such as "SDK", "Dev Guide", and "Reference" contain detailed information about different aspects of the Android development process.

Table of Contents

Before You Begin

Please read the following topics before you begin the course.

Minimum Hardware and Software Requirements

This is a hands-on programming course! You will be installing the Java Development Kit (JDK), Eclipse (IDE), Android SDK, and Android Development Tools (ADT) on your computer. Your computer must meet the following minimum requirements in order to run the programs:

Computer Hardware

Your computer must meet the following minimum specifications:

	Minimum
CPU	1.6GHz or faster processor
RAM	1024 MB
Display	1024 x 768 or higher resolution
Hard Disk Size	3GB available space
DVD Drive	DVD-ROM drive

Operating Systems

Your computer operating system must match one of the following:

Windows XP (x86) with Service Pack 3 or above (except Starter Edition)
Windows Vista (x86 and x64) with Service Pack 2 or above (except Starter Edition)
Windows 7 (x86 and x64)
Windows 8 or Windows 8 Pro (excluding Windows 8 RT)
Apple Mac OS X (10.6 or above)

You must also have an Internet connection to your computer in order to access online help, download and install software, and complete some of the activities in this course.

Conventions Used in This Text

This course will use certain styles (fonts, borders, etc.) to highlight text of special interest.

```
The source code will be in the 11-point Consolas font, in a single box like this.
```

Variable names will be in **`12-point Consolas bold`** text, similar to the way they will look in your development environment. For example: **`myVariable`**.

Function names, properties and keywords will be in **bold face** type, so that they are easily readable.

This picture highlights important concepts within a lesson.

Sidebars may contain additional information, tips, or background material.

This icon indicates a hands-on activity that you will complete on your computer.

What You Will Learn and Do In This Course

TeenCoder™: Android Programming will teach you the fundamentals of writing your own Android programs. You will be writing these programs using the Java programming language. This course is geared for high-school students who have expressed an interest in computer programming or who are looking for college-preparatory material.

You will learn to write *your own* mobile applications and begin to understand the building blocks for Android applications that you may use every day! Starting with the second chapter, you will complete a hands-on programming project at the end of each chapter. These projects will increase in complexity as you learn more about the Android environment.

What You Need to Know Before Starting

You must have completed the *TeenCoder™: Java Programming* course prior to starting this second semester course. The Java and object-oriented concepts taught in the first semester are prerequisites to learning and enjoying this Android programming material. You will need to be comfortable using the Eclipse development environment as taught in the first semester to write, build, run, and debug Java applications.

You are also expected to already know the basics of computer use before beginning this course. You need to know how to use the keyboard and mouse to select and run programs, use application menu systems, and work with the Windows or Mac OS operating system. You should understand how to store and load files on your hard disk, and how to use the Windows Explorer or Mac OS Finder to walk through your file system and directory structures. You should also have some familiarity with using text editors and using web browsers to find helpful information on the Internet.

Software Versions

You will be using the following free software in this course: *Java Development Kit (JDK)*, *Eclipse IDE for Java Developers*, the *Eclipse Android Development Tools (ADT)* plug-in and the *Android Software Developers Kit (SDK)*. Your course contains links to download and install instructions in PDF format on our website, http://www.HomeschoolProgramming.com. 3rd party websites may from time to time change their download process or release newer versions of their software. Our website will contain updated versions of the instructions as needed.

Getting Help

All courses come with a Solution Guide PDF and fully coded solutions for all activities. Simply install the "Solution Files" from your course setup program and you will be able to refer to the solutions as needed from the "Solution Menu". If you are confused about any activity you can see how we solved the problem! You may also contact us through the "Support" area of our website for further assistance.

Course Errata

We welcome your feedback regarding any course details that are unclear or that may need correction. Please contact us using our online "Getting Help" form. You can find a list of course errata for this edition on our website, http://www.HomeschoolProgramming.com.

Support for Multiple Operating Systems

This course was developed for use both on Microsoft Windows and Apple Mac OS X operating systems. The Java platform is compatible with both environments (and others). We will point out in text or by screen shots any differences between the operating systems. Where necessary, we will provide dedicated sets of instructions for handling each operating system. Be sure to follow the instructions that match the operating system you are using!

Directory Naming Conventions

On Windows operating systems, directory paths are traditionally represented with backslashes ("\") between folder names like this: "**TeenCoder\Java Programming**", but forward slashes ("/") work also. Mac OS directories use forward slashes as in "**TeenCoder/Java Programming**". In order to avoid cluttering the textbook with both representations, each time we specify a path, we will simply use one style. Be sure to change that style to match your operating system requirements if needed.

SAMPLE STUDENT LESSON

The following pages contain a sample student lesson from the TeenCoder: Android Programming textbook.

Chapter Nine: Displaying Images

So far we have shown you how to create many different types of Android controls. In this chapter, we will learn how to spice up our applications with image resources.

Lesson One: Adding Image Resources

Resources are components bundled with your program that are not part of your source code. You have already seen how to create layout and string resources in XML files underneath your projects's "res" directory. Your image resources will also go in folders underneath the "res" folder.

The Drawable Folders

Your Eclipse Package Explorer window shows you all of the files and folders in your project. Underneath your "res" directory you will notice four "drawable" folders. In Android, a *drawable* is any resource that can be drawn onto the screen. You will use these folders to manage image "drawable" resources. So why do we need more than one drawable directory? The answer lies in the sheer number and variety of different Android devices that are available on the market today.

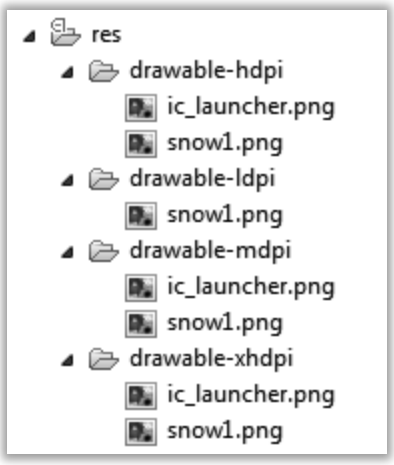
Your application may run on a device that is only 2 inches square or it may run on a tablet that is 10 inches or more in diagonal size. The Android OS will categorize the "size" of each screen as "small", "normal", large", or "extra large". In addition, devices may have different resolutions or screen densities. This is the measure of how many *pixels per inch* that a device can display. A low-density screen has fewer pixels per inch and does not have as clear a picture as that of a high-density screen. The screen density is also grouped by Android into "ldpi" (low density), "mdpi" (medium density), "hdpi" (high density), and "xhdpi" (extra-high density).

The following table shows the approximate physical size, density, and minimum width/height in "dp" of the "small", "normal", "large", and "extra large" device screen sizes. These values are *approximate* because new devices are released all the time, and the Android OS will make the decision internally what mode best matches a particular device.

Size	Density	Physical Size	Pixels Per Inch	Min Width (dp)	Min Height (dp)
Small	**ldpi**	2 – 4 inches	100 – 130	426	320
Normal	**mdpi**	3 – 5 inches	130 – 180	470	320
Large	**hdpi**	4 – 7 inches	180 – 260	640	480
Extra Large	**xhdpi**	7 – 10+ inches	260+	960	720

These different sizes and screen resolutions make designing an Android application a little bit trickier than a normal computer program! How do you use images that look good on small, low-density devices as well as larger, high-density devices? The answer is to create multiple images, one for each density.

You will notice four folders underneath "res\drawable" called "drawable-ldpi", "drawable-mdpi", "drawable-hdpi", and "drawable-xhdpi". These folders hold different density images to match target screens.



If you plan on supporting all four device screen densities, then you should provide different versions of each image that you want to use in your application. Each image should have the same name, but should be placed into the correct folder in the "/res" directory. In the example to the right, we have added a "snow1.png" image go each of the drawable folders. While this may seem like a lot of extra work in the beginning, making an application look great on any device is well worth the effort.

How do you actually create images with different densities? You can use image editing software that allows you to save the same file in different "pixels per inch" formats. There is no one accepted standard pixels-per-inch value for each density category, but good target densities are 120 ppi (ldpi), 160 ppi (mdpi), 240 ppi (hdpi), and 320 dpi (xhdpi). If you don't have different versions of your image, you can create a "res\drawable" folder without any density suffix. Or you can just place a single image in any one of the existing "drawable" folders such as "res\drawable-mdpi". Android will find whatever images are available and do the best it can to display them on each device.

If you do have different versions of each image, how do you figure out which image to display on the screen? Actually, you don't have to! Android will do the work for you. Once you have added your images to the correct folders, the Android system will choose which image looks the best on the current device.

Image Formats

Image files come in many different formats such as GIF, BMP, JPG, PNG, or TIFF. Android supports three of these formats. The preferred image format is a *Portable Network Graphics* or PNG file. Most image files will use *data compression* to squeeze a large amount of image into a smaller file size. A PNG image uses *lossless* data compression, which means it does not lose any image quality when the data is compressed. The result is a high-quality picture in a small image size. PNG images also support *transparency*, which is important when overlapping graphics on the screen.

You can also use JPEG or JPG files (short for *Joint Photographic Experts Group*), which are typically smaller in size than PNG files. JPG files do not support transparency, however, which limits their usefulness, especially in games. In addition, JPGs use a "lossy" data compression, so your image may lose some of the fine details or appear a bit blurry. JPG files are supported, but not widely used by Android applications.

The final image type that Android supports is the GIF or *Graphics Interchange Format* files. This is an older type of image that does support transparency, lossless data compression, and even animation. However GIF images can only contain 256 different colors and are therefore strongly discouraged on the Android device.

Image Resources

Just like any other resource, you will want to refer to an image in one of the "drawable" folders by an ID in your Java code and your XML files. Once an image is placed into any drawable folder and the program is compiled, you can use the standard "R" resource prefix followed by "drawable" and then the image name (without extension). Do not include the density, such as "hdpi"; just use the common filename that you placed in one or more drawable folders. This example refers to an image named "snow1.png":

```
R.drawable.snow1
```

The exact same ID refers to all of the versions of "snow1" that you have placed in any drawable folder.

In your XML layout files, you can refer to an image with the format "@drawable/<image filename>". Our same "snow1" picture would have this XML name:

```
"@drawable/snow1"
```

We'll talk about how to access and display these image resources in the next lesson.

Lesson Two: The ImageView Control

In the last lesson, you learned how to add image resources to your project. Now you'll discover how to display images on the screen using an **ImageView** control from the **android.widget** package. Let's say that we want to show a "drawable" image resource called "snowflake.png". The XML layout to create an **ImageView** control and display an image looks like this:

```
<ImageView
    android:id="@+id/myImageView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/snowflake" />
```

In addition to the standard id, width, and height properties, we are adding one new property: "android:src". This attribute selects the image to display by using the unique ID value. Remember for images stored in the drawable resource folders, the XML ID format is "@drawable/<image filename>".

Notice that we did not specify in which drawable directory our image is located. Hopefully, we have added a snowflake image in each of the supported densities. But if there is only one image, the Android will find it.

Scaling and Cropping with ImageView

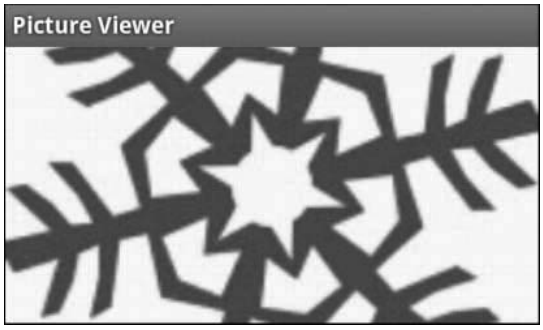
If we want more control over how the image is displayed in the control, we can use another attribute value called "android:scaleType". This attribute will control how the image is resized or adjusted to fit the size of the **ImageView** control. The scale type attribute can be set in the XML layout or in your Java code.

Java ImageView.ScaleType	XML Value	Description
MATRIX	"matrix"	Allows you to apply a 3D matrix of transforms for skewing, stretching, and rotating.
FIT_XY	"fitXY"	This value will scale the image to fit the height and width of the control. This will often distort the look of the image.
FIT_START	"fitStart"	Scales the image to fit entirely inside the available area while maintain the same aspect ratio (no distortion). Image will be aligned on the top and left edge of the display area.
FIT_CENTER	"fitCenter"	Scales the image to fit entirely inside the center of the available area while maintain the same aspect ratio (no distortion).
FIT_END	"fitEnd"	Scales the image to fit entirely inside the available area while maintain the same aspect ratio (no distortion). Image will be aligned on the bottom and right edge of the display area.

CENTER	"center"	Centers the image in the display area without any scaling
CENTER_CROP	"centerCrop"	Centers the image and scales it up without distortion to fill the entire area. Some parts may be cropped (cut) if they do not fit.
CENTER_INSIDE	"centerInside"	Centers the image and scales it up without distortion such that the largest dimension is completely filled.

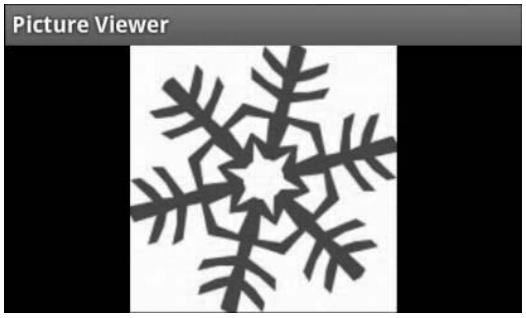
For example, here we have modified our <ImageView> with a width of "fill_parent" so the image will cover the entire width of the screen. Then we specified a "scaleType" of "centerCrop" to scale the image to fill the entire area, cropping one side or another as needed. You can see the snowflake was scaled up without distortion to cover the horizontal area completely, but some of the top and bottom were cut off.

```
<ImageView
    android:id= "@+id/myImageView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:src="@drawable/snowflake"
    android:scaleType="centerCrop"/>
```



Programmatically Managing the ImageView

You can also set the scale type attribute or manage other **ImageView** properties at runtime inside your Java code. Be sure to import the **android.widget.*** package and, of course, use the trusty **findViewById**() method to get a reference to the **ImageView** object by resource name.



Here we call **setScaleType**() on the **ImageView** and pass in the **ImageView.ScaleType. CENTER_INSIDE** value to scale and center the snowflake inside the area without cropping.

```
ImageView iv = (ImageView)findViewById(R.id.myImageView);
iv.setScaleType(ImageView.ScaleType.CENTER_INSIDE);
```

You can also change the image source at runtime as well by calling **setImageResource**() with a different ID:

```
iv.setImageResource(R.drawable.snow1);
```

Notice as you type "R.drawable." into your Eclipse IDE, you will see a handy pop-up list of all the available images in the drawable directories. That's just Eclipse trying to make your life easier!

SAMPLE SOLUTION GUIDE

The following pages contain sample solution material for an activity in the TeenCoder: Android Programming textbook.

Chapter Two Activity (Hello Android!)

Activity Description

In this activity, the student will demonstrate their understanding of the steps needed to create an Android program. They will be responsible for creating a basic Android project in the Eclipse software and then running the resulting program in the Android emulator on their computer.

The student should be able to follow the steps detailed in the textbook to complete this process. After they have run the program and viewed the default screen, they are asked to make a slight change to the "main.xml" file in their project.

Activity Output

When the program is first run, a screen like the following should appear on their computer:

After making some changes to the "main.xml" file, the student should run their program again and see a screen that looks like the following:

Code Required for Activity

The student will only be responsible for changing some code in the "**main.xml**" file for this project. The completed code will look something like this (the bold-face line notes the student's changes):

```
<?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="fill_parent"
      android:orientation="vertical" >

<TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Hello Bob!" />
</LinearLayout>
```

The completed project for this activity is located in the "Activity Solutions\Hello Android" folder underneath the Solution Files installation directory.