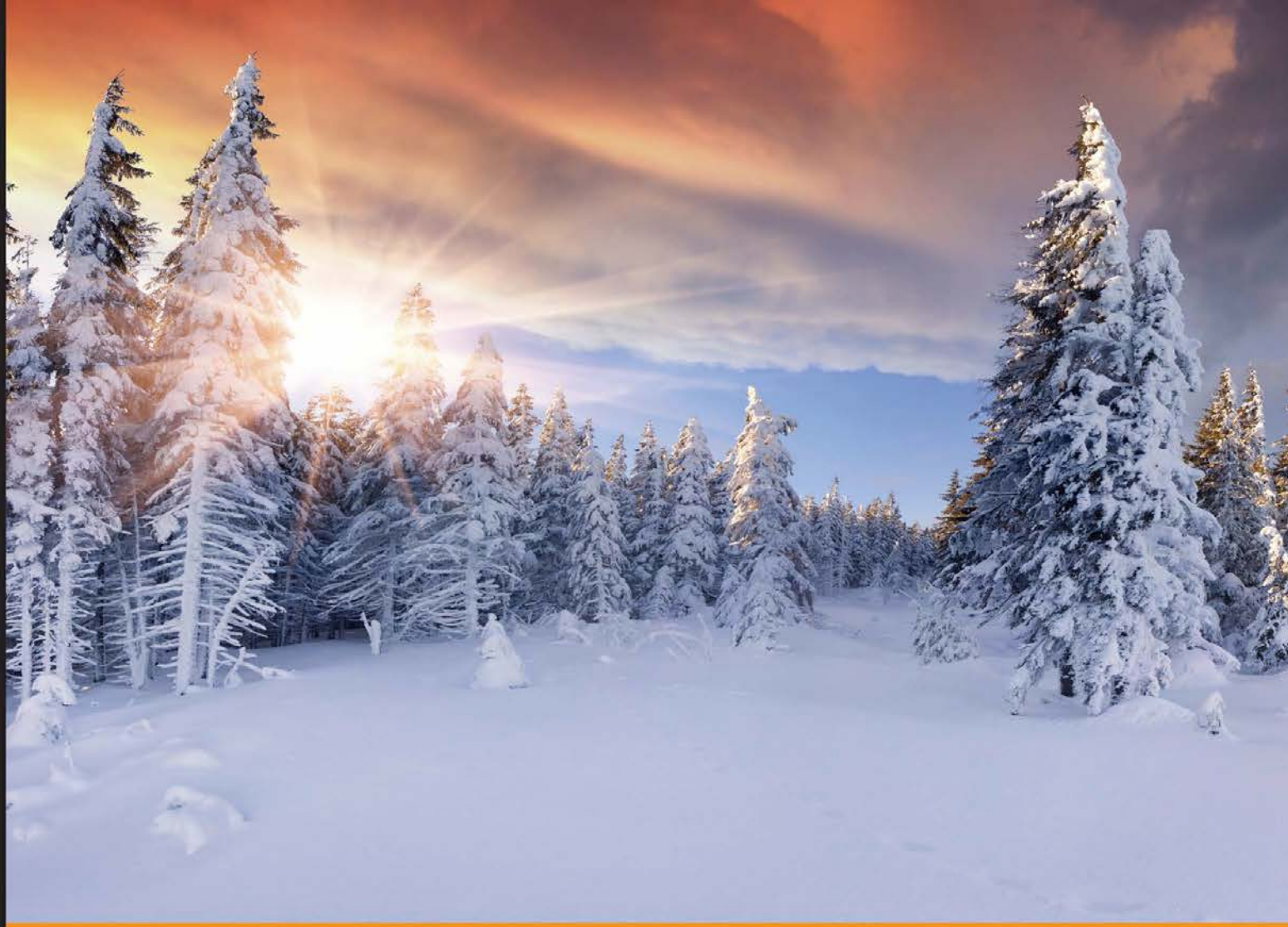

Community Experience Distilled

Learning Android Application Testing

Improve your Android applications through intensive testing and debugging

Paul Blundell
Diego Torres Milano

[PACKT] PUBLISHING open source* community experience distilled

Learning Android Application Testing

Improve your Android applications through intensive testing and debugging

Paul Blundell

Diego Torres Milano

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Learning Android Application Testing

First published: June 2011

Second edition: March 2015

Production reference: 1240315

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78439-533-9

www.packtpub.com

Credits

Authors

Paul Blundell

Diego Torres Milano

Reviewers

BJ Peter DeLaCruz

Noureddine Dimachk

Miguel L Gonzalez

Henrik Kirk

Sérgio Lima

João Trindade

Commissioning Editor

Taron Pereira

Acquisition Editor

Rebecca Youé

Content Development Editor

Manasi Pandire

Technical Editor

Indrajit A. Das

Copy Editors

Khushnum Mistry

Alfida Paiva

Vikrant Phadke

Adithi Shetty

Project Coordinator

Suzanne Coutinho

Proofreaders

Simran Bhogal

Joanna McMahon

Indexer

Hemangini Bari

Graphics

Valentina D'silva

Production Coordinator

Alwin Roy

Cover Work

Alwin Roy

About the Authors

Paul Blundell is an aspiring software craftsman and senior Android developer at Novoda. Before Novoda, he worked at AutoTrader and Thales, with apps that he released racking up over one million downloads. A strong believer in software craftsmanship, SOLID architecture, clean code, and testing, Paul has used this methodology to successfully nurture and create many Android applications. These include the Tesco launcher app, which was preinstalled for the recently released Hudl2 tablet; MUBI, a unique film streaming service; and the AutoTrader UK car search app.

If anyone wants to provide feedback, you can always tweet to him `@blundell_apps`. He also likes to write, so you can find more material at `http://blog.blundellapps.com/`.

I'd like to thank everyone at Novoda for being great guys/gals and helping each other all the time to learn and develop. Without the atmosphere of craftsmanship and constant learning, my skills and this book would not have been possible. Also, I'd like to thank my girlfriend for her endless patience. Every time she asked me to help her out, I'd give her the excuse of writing my book. Well, no more excuses because it is finished!

I'd like to acknowledge the legacy author of this book Diego Torres Milano for doing a great job. The chapters outlined are down to your insight into the world of testing on Android, and I hope my rewrite lives up to your ideals.

Finally, I'd like to thank all the people who don't know me but from whom I've learnt a lot. If you, as the reader, want a list of other authors for further research, this is it: Kent Beck, Martin Fowler, Robert C Martin, Romain Guy, Reto Meier, Mark Murphy, Eric Evans, Joshua Block, Ward Cunningham, Kevin Rutherford, JB Rainsberger, and Sandro Mancuso.

Diego Torres Milano has been involved with the Android platform since its inception, by the end of 2007, when he started exploring and researching the platform's possibilities, mainly in the areas of user interfaces, unit and acceptance tests, and Test-driven Development.

This is reflected by a number of articles mainly published on his personal blog (`http://dtmilano.blogspot.com`), and his participation as a lecturer in some conferences and courses, such as Mobile Dev Camp 2008 in Amsterdam (Netherlands) and Japan Linux Symposium 2009 (Tokyo), Droidcon London 2009, and Skillsmatter 2009 (London, UK). He has also authored Android training courses delivered to various companies in Europe.

Previously, he was the founder and developer of several open source projects, mainly CULT Universal Linux Thin Project (`http://cult-thinclient.sf.net`) and the very successful PXES Universal Linux Thin Client project (that was later acquired by 2X Software, `http://www.2x.com`). PXES is a Linux-based operating system specialized for thin clients, used by hundreds of thousands of thin clients all over the world. This project has a popularity peak of 35 million hits and 400K downloads from SourceForge in 2005. This project had a dual impact. Big companies in Europe decided to use it because of improved security and efficiency; and organizations, institutions, and schools in some development countries in South America, Africa, and Asia decided to use it because of the minimal hardware requirements, having a huge social impact of providing computers, sometimes recycled ones, to everyone.

Among the other open source projects that he founded are Autoglade, Gnome-tla, and JGlade, and he has contributed to various Linux distributions, such as RedHat, Fedora, and Ubuntu.

He has also given presentations at the LinuxWorld, LinuxTag, GUADEC ES, University of Buenos Aires, and so on.

Diego has also developed software, participated in open source projects, and advised companies worldwide for more than 15 years.

He can be contacted at `dtmilano@gmail.com`.

About the Reviewers

BJ Peter DeLaCruz graduated with a master's degree in computer science from the University of Hawaii at Manoa. In 2011, he began his career as a software developer at Referentia Systems Inc. in Honolulu, Hawaii. At Referentia, he assisted in the development of the LiveAction product. After working at Referentia for 2.5 years, he was hired as a Java web developer by the University of Hawaii. Between fall 2014 and spring 2015 semesters, he upgraded Laulima (`http://laulima.hawaii.edu`), the learning management system that the university uses for traditional face-to-face, online, and hybrid classes.

BJ holds three Java certifications, including the Oracle Certified Master, Java SE 6 Developer certification.

He is a successful Android developer. As of January 2015, he has published seven Android apps on Google Play. His latest app, Chamorro Dictionary, is an excellent learning tool for the Chamorro language. You can check out his apps at `http://tinyurl.com/google-play-bpd`.

BJ really likes Gradle because it makes building applications very easy. He was a reviewer for *Gradle in Action*.

His hobbies include learning the Japanese language, reading books about Japanese culture, and making YouTube videos. You can contact him at `bj.peter.delacruz@gmail.com`. You can also visit his website at `http://www.bjpeter.com`.

I want to thank God for giving me the opportunity to review this book. I also want to thank Nikita Michael for inviting me to become a reviewer and Suzanne Coutinho for sending all the chapters to review. Arigatou gozaimasu!

Noureddine Dimachk is a passionate video gamer since birth. Noureddine started building games using The Games Factory when he was just 10 years old.

Today, he leads a multinational team of 17 enthusiastic developers spread across Lebanon, Argentina, and India to build cutting-edge applications that serve millions of concurrent GSM subscribers, in addition to mobile applications.

A geek by nature, Noureddine likes to experiment with new technologies in his spare time, and he's a passionate *Dota 2* player.

I would like to thank my amazing wife for standing by me and supporting me in my technical ventures.

Miguel L Gonzalez is a Spanish software engineer working in the United Kingdom since 2010. He took his first programming course at the early age of eight, and it has been his main passion and hobby since then. He soon became attracted to the Web and Internet, which lead him to study telecommunications engineering.

He has worked as a researcher in the university, designing accessible hardware and wireless sensor networks, teaching web development, developing a mixture of Java hardware, desktops, and web apps, and is the head of development in an agency. Since the time he arrived in the UK, he has mainly focused on web and native development for mobiles, and he developed a few Android and iOS apps in coANDcoUK. In 2013, he joined BBC to work on iPlayer, BBC's catch-up service. It was here that he became more serious about unit testing, behavioral testing, and how to drive success via continuous integration.

He tries to keep improving his projects, which can be found at `http://github.com/ktzar` and maintain his personal website, `http://mentadreams.com`. Since his son Alex was born, the spare time for side projects has been reduced, but his wife, Dalia, helps him to find time for them. Nevertheless, he's looking forward to playing *Monkey Island*, designing games, playing the guitar, and traveling the world with his offspring in a few years time.

Henrik Kirk holds a master's degree in computer science from Aarhus University and has over 5 years of experience in Android application development. He is curious about new technologies and has been using Scala as well as Java for Android development. He also enjoys optimizing the user experience through speed and responsive design. He is currently employed as the lead developer at Lapio, creating an awesome timing and race experience for athletes in the US and Europe. In his spare time, he races his mountain bike.

Sérgio Lima is a software engineer and an airplane pilot. It's easy to see that he's a very ambitious person with broad and, at the same time, specific interests. He currently works at a Portuguese company that aims to revolutionize the world with telecom and mobile applications. His curriculum started with a master's degree in electronics and telecommunications and he specialized in computer programming and computer vision. After working at some institutions in Portugal, he worked at CERN in Switzerland, before returning to his home country.

He also loves to fly small planes, such as the Piper "Cherokee" and "Tomahawk", from the nearby aerodrome, to see Portugal from above, admire the radiant sceneries of the country, and experience the freedom of flying.

I would like to thank my family and specially my wonderful princess, "Kika", for her patience, support, and love during the process of reviewing this book.

João Trindade is a software developer who specializes in developing Android apps.

Currently, he is part of a startup in Milan that tracks your mobile phone usage and suggests the best tariff plan for your needs.

He completed his PhD in computer engineering at Lisbon Tech and is interested in everything related to mobile development, software testing, docker containers, or cloud computing.

For 6 years he was a researcher involved in multiple international research projects and has published 18 peer reviewed articles.

His twitter handler is @joaotrindade and his personal web page is `http://joaoptrindade.com`.

He contributes to various open source products on GitHub. You can see his profile at `http://github.com/joninvski`.

www.PacktPub.com

Support files, eBooks, discount offers, and more

For support files and downloads related to your book, please visit `www.PacktPub.com`.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`https://www2.packtpub.com/books/subscription/packtlib`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

Why subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

Free access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

It doesn't matter how much time you invest in Android design, or even how careful you are when programming, mistakes are inevitable and bugs will appear. This book will help you minimize the impact of these errors in your Android project and increase your development productivity. It will show you the problems that are easily avoided, to help get you quickly to the testing stage.

Android Application Testing Guide is the first and only book providing a practical introduction to the most commonly available techniques, frameworks, and tools to improve the development of your Android applications. Clear, step-by-step instructions show how to write tests for your applications and assure quality control using various methodologies.

The author's experience in applying application testing techniques to real-world projects enables him to share insights on creating professional Android applications.

The book covers the basics of framework support for tests to architectures and techniques such as Test-driven Development, which is an agile component of the software development process and a technique where you will tackle bugs early on. From the most basic unit tests applied to a sample project to more sophisticated performance tests, this book provides a detailed description of the most widely used techniques in the Android testing world in a recipe-based approach.

The author has extensive experience of working on various development projects throughout his professional career. All this research and knowledge has helped create a book that will serve as a useful resource to any developer navigating the world of Android testing.

What this book covers

Chapter 1, *Getting Started with Testing*, introduces the different types of testing and their applicability to software development projects in general and to Android in particular. It then goes on to cover testing on the Android platform, unit testing and JUnit, creating an Android test project and running tests.

Chapter 2, *Understanding Testing with the Android SDK*, starts digging a bit deeper to recognize the building blocks available to create the tests. It covers Assertions, TouchUtils, which are intended to test user interfaces, mock objects, instrumentation, and TestCase class hierarchies.

Chapter 3, *Baking with Testing Recipes*, provides practical examples of different situations you will commonly encounter while applying the disciplines and techniques described before. The examples are presented in a cookbook style so you can adapt and use them for your projects. The recipes cover Android unit tests, activities, applications, databases and ContentProviders, services, UIs, exceptions, parsers, memory leaks, and a look at testing with Espresso.

Chapter 4, *Managing Your Android Testing Environment*, provides different conditions to run the tests. It starts with the creation of the Android Virtual Devices (AVD) to provide different conditions and configurations for the application under test and runs the tests using the available options. Finally, it introduces monkey as a way to generate simulated events used for testing.

Chapter 5, *Discovering Continuous Integration*, introduces this agile technique for software engineering and automation that aims to improve the software quality and reduce the time taken to integrate changes by continuously applying integration and testing frequently.

Chapter 6, *Practicing Test-driven Development*, introduces the Test-driven Development discipline. It starts with a general revision and later on moves to the concepts and techniques closely related to the Android platform. This is a code-intensive chapter.

Chapter 7, *Behavior-driven Development*, introduces Behavior-driven Development and some concepts, such as the use of a common vocabulary to express the tests and the inclusion of business participants in the software development project.

Chapter 8, *Testing and Profiling Performance*, introduces a series of concepts related to benchmarking and profiles from traditional logging statement methods to creating Android performance tests and using profiling tools.

Chapter 9, *Alternative Testing Tactics*, covers adding code coverage to ensure you know what is tested and what isn't, as well as testing on the host's Java Virtual Machine, investigating Fest, Spoon, and the future of Android testing to build upon and expand your Android testing range.

What you need for this book

To be able to follow the examples in the different chapters, you need a common set of software and tools installed and several other components that are described in every chapter in particular, including their respective download locations.

All the examples are based on the following:

- Mac OSX 10.9.4, fully updated
- Java SE version 1.6.0_24 (build 1.6.0_24-b07)
- Android SDK tools, revision 24
- Android SDK platform-tools, revision 21
- SDK platform Android 4.4, API 20
- Android support library, revision 21
- Android Studio IDE, Version: 1.1.0
- Gradle version 2.2.1
- Git version 1.8.5.2

Who this book is for

If you are an Android developer looking to test your applications or optimize your application development process, then this book is for you. No previous experience in application testing is required.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "To invoke the `am` command we will be using the `adb shell` command".

A block of code is set as follows:

```
dependencies {
    compile project(':dummylibrary')
}
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
fahrenheitEditNumber
.addTextChangedListener(
newFehrenheitToCelciusWatcher(fahrenheitEditNumber,
  celsiusEditNumber));
}
```

Any command-line input or output is written as follows:

```
junit.framework.ComparisonFailure: expected:<[]> but was:<[123.45]>
at com.blundell.tut.EditNumberTests.testClear(EditNumberTests.java:31)
at java.lang.reflect.Method.invokeNative(Native Method)
at android.test.AndroidTestRunner.runTest(AndroidTestRunner.java:191)
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "The first test performs a click on the **Go** button of the Forwarding Activity."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files from your account at `http://www.packtpub.com` for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Getting Started with Testing

Firstly, I will avoid introductions to Android since it is covered in many books already, and I am inclined to believe that if you are reading a book that covers this more advanced topic, you will have already started with Android development.

I will be reviewing the main concepts behind testing, and the techniques, frameworks, and tools available to deploy your testing strategy on Android.

After this overview, we can put the concepts learned into practice. In this chapter we will cover:

- Setting up the infrastructure to test on Android
- Running unit tests using JUnit
- Creating an Android instrumentation test project
- Running multiple tests

We will be creating a simple Android project and its companion tests. The main project will be bare bones so that you can concentrate on the testing components.

I would suggest that new developers with no Android testing experience read this book. If you have more experience with Android projects and have been using testing techniques for them, you might read this chapter as a revision or reaffirmation of the concepts.

Why, what, how, and when to test?

You should understand that early bug detection saves a huge amount of project resources and reduces software maintenance costs. This is the best known reason to write tests for your software development project. Increased productivity will soon be evident.

Additionally, writing tests will give you a deeper understanding of the requirements and the problem to be solved. You will not be able to write tests for a piece of software you don't understand.

This is also the reason behind the approach of writing tests to clearly understand legacy or third-party code and having the testing infrastructure to confidently change or update the codebase.

The more the code is covered by your tests, the higher the likelihood of discovering hidden bugs.

If, during this coverage analysis, you find that some areas of your code are not exercised, additional tests should be added to cover this code as well.

To help in this request, enter Jacoco (`http://www.eclemma.org/jacoco/`), an open source toolkit that measures and reports Java code coverage. It supports various coverage types, as follows:

- Class
- Method
- Block
- Line

Coverage reports can also be obtained in different output formats. Jacoco is supported to some degree by the Android framework, and it is possible to build a Jacoco instrumented version of an Android app.

We will be analyzing the use of Jacoco on Android to guide us to full test coverage of our code in *Chapter 9*, *Alternative Testing Tactics*.

This screenshot shows how a Jacoco code coverage report is displayed as an HTML file that shows green lines when the code has been tested:

core > com.blundell.tut > TemperatureConverter.java

TemperatureConverter.java

```
1.  package com.blundell.tut;
2.
3.  public final class TemperatureConverter {
4.
5.      static final double ABSOLUTE_ZERO_C = -273.15d;
6.      static final double ABSOLUTE_ZERO_F = -459.67d;
7.
8.      private static final String ERROR_MESSAGE_BELOW_ZERO_FMT = "Invalid temperature: %.2f%c below absolute zero";
9.
10.     private TemperatureConverter() {
11.         // non-instantiable helper class
12.     }
13.
14.     public static double fahrenheitToCelsius(double fahrenheit) {
15.         if (fahrenheit < ABSOLUTE_ZERO_F) {
16.             String msg = String.format(ERROR_MESSAGE_BELOW_ZERO_FMT, fahrenheit, 'F');
17.             throw new InvalidTemperatureException(msg);
18.         }
19.         return ((fahrenheit - 32) / 1.8d);
20.     }
21.
22.     public static double celsiusToFahrenheit(double celsius) {
23.         if (celsius < ABSOLUTE_ZERO_C) {
24.             String msg = String.format(ERROR_MESSAGE_BELOW_ZERO_FMT, celsius, 'C');
25.             throw new InvalidTemperatureException(msg);
26.         }
27.         return (celsius * 1.8d + 32);
28.     }
29.
30. }
```

By default, the Jacoco gradle plugin isn't supported in Android Studio; therefore, you cannot see code coverage in your IDE, and so code coverage has to be viewed as separate HTML reports. There are other options available with other plugins such as Atlassian's Clover or Eclipse with EclEmma.

Tests should be automated, and you should run some or all tests every time you introduce a change or addition to your code in order to ensure that all the conditions that were met before are still met, and that the new code satisfies the tests as expected.

This leads us to the introduction of **Continuous Integration**, which will be discussed in detail in *Chapter 5*, *Discovering Continuous Integration*, enabling the automation of tests and the building process.

If you don't use automated testing, it is practically impossible to adopt Continuous Integration as part of the development process, and it is very difficult to ensure that changes would not break existing code.

Having tests stops you from introducing new bugs into already completed features when you touch the code base. These regressions are easily done, and tests are a barrier to this happening. Further, you can now catch and find problems at compile time, that is, when you are developing, rather than receiving them as feedback when your users start complaining.

What to test

Strictly speaking, you should test every statement in your code, but this also depends on different criteria and can be reduced to testing the main path of execution or just some key methods. Usually, there's no need to test something that can't be broken; for example, it usually makes no sense to test getters and setters as you probably won't be testing the Java compiler on your own code, and the compiler would have already performed its tests.

In addition to your domain-specific functional areas that you should test, there are some other areas of an Android application that you should consider. We will be looking at these in the following sections.

Activity lifecycle events

You should test whether your activities handle lifecycle events correctly.

If your activity should save its state during the `onPause()` or `onDestroy()` events and later be able to restore it in `onCreate(Bundle savedInstanceState)`, then you should be able to reproduce and test all these conditions and verify that the state was correctly saved and restored.

Configuration change events should also be tested as some of these events cause the current Activity to be recreated. You should test whether the handling of the event is correct and that the newly created Activity preserves the previous state. Configuration changes are triggered even by a device rotation, so you should test your application's ability to handle these situations.

Database and filesystem operations

Database and filesystem operations should be tested to ensure that the operations and any errors are handled correctly. These operations should be tested in isolation at the lower system level, at a higher level through `ContentProviders`, or from the application itself.

To test these components in isolation, Android provides some mock objects in the `android.test.mock` package. A simple way to think of a mock is as a drop-in replacement for the real object, where you have more control of the object's behavior.

Physical characteristics of the device

Before shipping your application, you should be sure that all of the different devices it can be run on are supported, or at least you should detect the unsupported situation and take pertinent measures.

The characteristics of the devices that you should test are:

- Network capabilities
- Screen densities
- Screen resolutions
- Screen sizes
- Availability of sensors
- Keyboard and other input devices
- GPS
- External storage

In this respect, an Android emulator can play an important role because it is practically impossible to have access to all of the devices with all of the possible combinations of features, but you can configure emulators for almost every situation. However, as mentioned before, leave your final tests for actual devices where the real users will run the application so you get feedback from a real environment.

Types of tests

Testing comes in a variety of frameworks with differing levels of support from the Android SDK and your IDE of choice. For now, we are going to concentrate on how to test Android apps using the instrumented Android testing framework, which has full SDK and ASide support, and later on, we will discuss the alternatives.

Testing can be implemented at any time in the development process, depending on the test method employed. However, we will be promoting testing at an early stage of the development cycle, even before the full set of requirements has been defined and the coding process has been started.

There are several types of tests depending on the code being tested. Regardless of its type, a test should verify a condition and return the result of this evaluation as a single Boolean value that indicates its success or failure.