

## Module Guide

<b>Faculty</b>	Information Technology		
<b>Module Code</b>	ITJA321	<b>Module Name</b>	Java and Distributed Systems
<b>NQF Level</b>	7	<b>Credit Value</b>	15
<b>Semester</b>	2/2020	<b>Year Level</b>	3
<b>Module Leader</b>	Mr Sheunesu Makura	<b>Copy Editor</b>	Mr Kevin Levy
<b>Lecturing Hours</b>	Refer to time allocation	<b>Tutorial/Practical Hours</b>	N/A
<b>Notional Hours</b>	150	<b>Prerequisites</b>	ITJA211

The module guide must be read in conjunction with the prescribed textbook. This document will be the first port of call to understanding what will be assessed and which assessments form part of the module.

The purpose of the module guide is to highlight:

- The learning outcomes and assessment criteria that need to be met to pass the module
- The assessment required to be completed for the module
- The additional resources required for the module
- The topics that will be focused on for the module

### Module Aim

The aim of this module is to enable students to understand computer networks and distributed computer systems while applying theoretical knowledge to practical scenarios. Skills obtained during this module could be equally applied to other fields such as game theory or artificial intelligence relating to the design and development of a scalable system.

## Module Description

As an IT professional, one has to be able to create a distributed system which consists of linking a collection of autonomous computers to a computer network and equipping such with distributed system software. In order to do this effectively, one needs to understand how distributed systems work.

Distributed computing systems allow corporations with various computer systems access to a networking system that allows resource sharing, openness, concurrency, scalability, fault tolerance and transparency. Additionally, a distributed system enables the use of any hardware, software or data anywhere in the system.

## Teaching and Learning Methodology (Blended Learning)

A blended approach to learning will be implemented. This will involve both (1) online learning in the form of live and recorded lectures, as well as incorporating multimedia tools to ensure maximum accessibility for all students; and (2) contact lecture sessions, where applicable (practical sessions/skills classes/special classes) lectures will take place in the form of open/distance learning and will be conducted via online platforms, allowing for active student engagement and discussion. In addition, pre-recorded lectures/lessons which aim to go through and explain module-related concepts will be available on *myLMS*. Some lecturers may also include other forms of multimedia as part of online learning to further enrich your knowledge on particular concepts and theories. The aim is to accommodate every student by incorporating a blended learning approach which will assist in moving from face-to-face engagement towards online learning to ensure that every student has access to everything that they need to succeed in their studies.

## Learning Time Allocation

Mode	Learning Activity	Hours per Week	Hours per Semester	Hours%
Online Sessions (Discussions)	Distance technology driven lectures (live or recorded) or workshop attendance according to a pre-determined schedule	2	32	21.4
Pre-recorded lectures	Forum activities	2.5	39.3	26.2
Group and individual consultations	Self-study	2.5	39.3	26.2
Distance	Case studies/research/ practical activities	2.5	39.3	26.2
<b>TOTAL (100%) = 150 hours</b>			<b>150</b>	<b>100</b>

## Learning Outcomes

By the end of this module, you will be able to:

Learning Outcomes	Assessment Criteria
1. Comply with the goals, architecture and changes of distributed systems.	1.1 Understand the concept of 'distributed computing' 1.2 Know the advantages and limitations of distributed computing 1.3 Understand architecture and the various languages used to implement distributed computing Blended learning activity: • Exercises using NetBeans IDE/Android Studio IDE
2. Apply the concept of 'parameter passing' of Java objects	2.1 Understand single- and multidimensional arrays 2.2 Use selections, loops and methods. Define classes and objects 2.3 Understand inheritance and polymorphism

	<p>Blended learning activity:</p> <ul style="list-style-type: none"> <li>Exercises using NetBeans IDE/Android Studio IDE</li> </ul>
3. Explore the relationships among Java Collection Framework and the ability to traverse a Collection	<p>3.1 Traverse and create instances of the Java Collection Framework interface</p> <p>3.2 Differentiate between HashSet, LinkedSet and TreeSet</p> <p>3.3 Know when to use ArrayList rather than LinkedList</p> <p>3.4 Understand the relationship between HashMap, Map, Vector, Queue and Priority Queue</p> <p>Blended learning activity:</p> <ul style="list-style-type: none"> <li>Exercises using NetBeans IDE/Android Studio IDE</li> </ul>
4. Construct concurrent applications using multiple threads	<p>4.1 Know how to develop task classes</p> <p>4.2 Understand thread synchronisation</p> <p>4.3 Know how to avoid deadlocks</p> <p>4.4 Understand the life-cycle of a thread</p> <p>Blended learning activity:</p> <ul style="list-style-type: none"> <li>Exercises using NetBeans IDE/Android Studio IDE</li> </ul>
5. Develop simple distributed applications using Java networking capabilities	<p>5.1 Use sockets on both client and server ends</p> <p>5.2 Develop servers for multiple clients</p> <p>5.3 Send and receive objects on a network</p> <p>Blended learning activity:</p> <ul style="list-style-type: none"> <li>Exercises using NetBeans IDE/Android Studio IDE</li> </ul>
6. Create a simple database application in Java using JDB	<p>6.1 Load database drivers and execute SQL statements</p> <p>6.2 Use the PreparedStatement</p> <p>6.3 Use the CallableStatement</p>

	Blended learning activity: <ul style="list-style-type: none"> <li>Exercises using NetBeans IDE/Android Studio IDE</li> </ul>
7. Build a distributed application using remote method invocation	7.1 Differentiate between an RMI stub and skeleton 7.2 Define and implement an RMI interface 7.3 Create distributed objects 7.4 Register distributed objects with RMI
	Blended learning activity: <ul style="list-style-type: none"> <li>Exercises using NetBeans IDE/Android Studio IDE</li> </ul>

## Prescribed Resource(s)

### Textbook(s)

Liang, Y.D. 2018. *Introduction to Java programming and Data Structures*. 11th edition. Pearson.  
 ISBN: 9781292221878

The following resource(s) will be made available on *myLMS*, which you must check regularly:

- Assignment specification
- Blended learning items
- Continuous assessments
- Exam scopes
- Important notifications from your lecturer
- Module guide
- Module announcements

## Recommended Resource(s)

Take note that all disciplines and their corresponding textbooks are frequently updated. Therefore, you should use the latest editions, where available. Recommended resources should be used for research purposes. There is a range of resources related to this module, including the following:

## **Textbook(s)**

Cadenhead, R. 2020. *Sams Teach Yourself Java in 21 Days*. Eighth Edition

Deitel, P. & Deitel, H. 2017. *Java: How to Program Late Objects:11th edition*. Pearson.

DiMarzio, J.F. (2017) *Android Programming with Android Studio, Fourth Edition*. John Wiley and Sons, Inc. ISBN:978-1-118-70559-9.

Eckel, B. 2006. *Thinking in Java*. 4th edition. Prentice Hall.

Morrison, M. 2006. *Sams Teach Yourself XML in 24 hours*: Sams Publishing

Steele, J. and To, N. 2013. *The Android Developers Cookbook: Building Applications with the Android SDK* (2nd Edition). Pearson.

## **Video(s)**

Weeks 11 - 13: Coding in Flow, CF. 2018. SQLITE Android Tutorials. [Online] Available at: <https://www.youtube.com/playlist?list=PLrnPJCHvNZuAPyh6nRXsvf5hF48SJWdJb> . [Accessed: 15 May 2020].

Weeks 11-13: Kamran Bhatti, KB. 2016. Android Insert Data into MySql Database Server - Android Mysql Connection. [Online] Available at: <https://www.youtube.com/watch?v=ryMj8xnZkSQ> . [Accessed: 15 May 2020].

## **Online Document(s)**

Eck, D.J. 2019. *Introduction to Programming Using Java, Eighth Edition* [Online] Available at: <http://math.hws.edu/javanotes/> [Accessed: 14 May 2020].

## **Website(s)**

Web pages provide access to a further range of internet information sources. Lecturers may download the web-related material for you to access offline. You must use this resource with care, justifying the use of information gathered.

The Java Tutorials [Online] Available at: <http://docs.oracle.com/javase/tutorial/> [Accessed: 14 May 2020].

Oracle Technology Network [Online] Available at:

<https://www.oracle.com/technical-resources/> [Accessed: 14 May 2020].

## Supporting Documents

Geyer, L., Levin, A., Makati, P., Pierce, R., Potter, M., and Wheeler, A. 2019. *PIHE Guide to Referencing (Harvard Referencing Method)*. Unpublished document. Pearson Institute of Higher Education

## Essential Requirements

- Access to the institution's Learner Management Systems(*myLMS*) to access all study material.
- Access to Microsoft Teams to attend online lectures and consultations.
- Access to a resource centre or an online library with a wide range of relevant resources including textbooks, newspaper articles, journal articles, organisational publications and databases.
- Access to a range of academic journals in electronic format via ProQuest or other databases.

## ICT Requirements

ICT Required	Reason	Lecture Week(s)
Computer with Internet installed with the following: <ul style="list-style-type: none"><li>• NetBeans 8.2</li><li>• Java JDK 8</li><li>• XAMPP 7.4</li><li>• Android Studio 3.6</li></ul>	Take Home Test Project Take Home Assessment	1 - 13

## Formative Assessment(s)

You may be required to complete some assessments online on *myLMS*. The following guidelines may apply:

- You may be required to write an online assessment directly on *myLMS*. You will need to log in at a specified time to attempt the assessment. Once the allocated time expires, the assessment will no longer be available to complete.
- Time limits should be checked before commencing assessments. Refer to the Assessment Details table of this module guide.
- Ensure that your internet connection is stable.

- In some cases, assessments are not available indefinitely and will only be available for a day or two.
- The marks for some assessments may only be available (with a memorandum) after all students have attempted the assessment after the assessment due date.

It is the students' responsibility to retrieve their assessment feedback and verify their marks on the day they are available. No adjustment of marks will be entertained beyond one week after marks were released.

### **Test(s)**

There will be **three** take home tests for this module and will count 20% towards the final mark.

If a test is missed because of illness, a doctor's note must be presented within 48 hours of the missed test to the academic manager/administrator/coordinator.

To make up for this missing assessment, you may be able to write a deferred test. However, in order to gain entry to this test, you will have to follow various procedures and meet certain criteria. You must complete a *Deferred Test Application Form* available on *myLMS*. You will be required to pay a non-refundable fee per application. Each test missed requires a separate application. This will be your only opportunity to make up for a missed test.

### **Project(s)**

There is only **one** project for this module. The project will be completed individually. The project is Android based. In order for students to achieve a pass mark, they should spend approximately 10 to 15 hours working on the project. The project will count 10% towards the final mark.

Assignments and projects must be submitted on or before the due date to the lecturer in class or as per arrangement. Five percent (5%) will be deducted for every day that the assignment or project is late, up to a maximum of three days. Assignments and projects that are more than three days late will be awarded a zero.

### **Take Home Assessment**

There will be one take home assessment written at the end of the semester. The assessment will count 50% towards the final mark.



## Continuous Assessments

Continual formative assessment may be conducted so that you are given feedback on your progress in the achievement of specific learning outcomes. The formative assessment tasks may take the form of one of the following:

- A five-item multiple-choice test
- A short-questions test
- Construction of concept maps
- Take-home tests with long questions
- Short practical tasks
- Short class/online presentations

Your lecturers will stipulate the date(s) of these assessments and scope of coverage.

## Plagiarism

All assignments and reports must be submitted to the online similarity checker (Turnitin) available on *myLMS* prior to being submitted for marking. When submitting your assignment/report, it is compulsory to submit the entire Turnitin report. Marks will be deducted in accordance with the institutional policy.

Also, when submitting assessments, you should include the applicable completed and signed assessment coversheet as an acknowledgement that the work submitted is your own original work, except for source material explicitly acknowledged. This declaration will serve as proof that you are aware of the Institution's policies and regulations on academic integrity.

## Final Mark

In order to pass the module a final average of 50% or higher is required for the entire module.

The final mark is calculated as follows:

**Final Mark** [(Continuous assessment percentage × 0.10) + (Project percentage × 0.20)  
+ (Take home test percentage × 0.20) + (Take home assessment percentage × 0.50)]

## Details of Assessments

Methods of Assessment	Weighting <sup>1</sup>	Dates
<b>Semester 2</b>		
Project	20%	05/10/2020 – 09/10/2020 Scope of coverage: Weeks 7, 10 - 13
Take Home Test 1	20%	03/08/2020 – 07/08/2020 Scope of coverage: Weeks 1 – 4 Availability: 03/08/2020 at 00:00 – 04/08/2020 at 23:59 Duration of timed assessments: 48 Hour(s)
Take Home Test 2		24/08/2020 - 28/08/2020 Scope of coverage: Weeks 5 – 7 Availability: 24/08/2020 at 00:00 – 25/08/2020 at 23:59 Duration of timed assessments: 48 Hour(s)
Take Home Test 3		21/09/2020 - 25/09/2020 Scope of coverage: Weeks 8 – 10 Availability: 21/09/2020 at 00:00 – 22/09/2020 at 23:59 Duration of timed assessments: 48 Hour(s)
Take Home Assessment	50%	19/10/2020 – 23/10/2020 Scope of coverage: Weeks 1 – 13 Availability: 19/10/2020 at 00:00 – 20/10/2020 at 23:59 Duration of timed assessments: 48 Hour(s)

<sup>1</sup> Refer to the **Conditions of Enrolment**, available on *myLMS*.

Continuous Assessment	10%	Lecturer will stipulate the date(s) of these assessments and scope of coverage.
All formative marks submitted		2: 04/11/2020

## Putting Together a Portfolio of Evidence

You must demonstrate, through the presentation of evidence, that you have met all module requirements within the qualification being undertaken. To do this, you must organise your evidence into what is known as a 'portfolio'.

A portfolio will take time and effort to complete. It is a means of focusing and demonstrating to others your strengths and achievements. A portfolio is an important resource that you may find useful to retain once you have achieved your qualification, particularly when applying for future positions.

You are encouraged to read more about building a portfolio and to begin populating your evidence to illustrate your full skill set to future employers.

## Consultations

Lecturers will be available for consultation. Specific details in this regard will be made available on your campus specific *myLMS* pages. You must give lecturers 24 hours' notice for appointments. Consultation meetings can be requested via email. It is important that you detail the requirements (chapter, section, etc.) for your consultation.

## Module Content

You are required to attend all classes. In addition, exercises and activities, which are supplied by lecturers, are compulsory.

Continuous assessments may run throughout the semester.

## Semester 2: Schedule

Lecture Weeks	Topics and Assessment Criteria Covered	Assessments	References
---------------	--	-------------	------------

<b>1</b> 2: 06/07/2020 – 10/07/2020	Introduction to Distributed Computing AC: 1.1, 1.2, 1.3		<ul style="list-style-type: none"> <li>• Appendix A</li> </ul>
<b>2</b> 2: 13/07/2020 – 17/07/2020	Java Collection Framework AC: 3.1, 3.2		<ul style="list-style-type: none"> <li>• Chapter 20, 21</li> </ul>
<b>3</b> 2: 20/07/2020 – 24/07/2020	Java Collection Framework AC: 3.1, 3.2		<ul style="list-style-type: none"> <li>• Chapter 20, 21</li> </ul>
<b>4</b> 2: 27/07/2020 – 31/07/2020	Multithreading AC: 4.1, 4.2, 4.3, 4.4		<ul style="list-style-type: none"> <li>• Chapter 32</li> </ul>
<b>5</b> 2: 03/08/2020 – 07/08/2020	Networking AC: 5.1, 5.2	Take Home Test 1	<ul style="list-style-type: none"> <li>• Chapter 33</li> </ul>
<b>6</b> 2: 11/08/2020 – 14/08/2020	Networking AC: 5.1, 5.2		<ul style="list-style-type: none"> <li>• Chapter 33</li> </ul>
<b>7</b> 2: 17/08/2020 – 21/08/2020	Advanced Java Database Programming AC: 6.1, 6.2, 6.3		<ul style="list-style-type: none"> <li>• Chapter 34 - 35</li> </ul>
<b>8</b> 2: 24/08/2020 – 28/08/2020	Remote Method Invocation AC: 7.1, 7.2, 7.3, 7.4	Take Home Test 2	<ul style="list-style-type: none"> <li>• Chapter 40</li> <li>• Appendix B</li> </ul>
2: 31/08/2020 – 04/09/2020	<b>Semester Break</b>		
<b>9</b> 2: 07/09/2020 – 11/09/2020	Remote Method Invocation AC: 7.1, 7.2, 7.3, 7.4		<ul style="list-style-type: none"> <li>• Chapter 40</li> <li>• Appendix B</li> </ul>
<b>10</b> 2: 14/09/2020 – 18/09/2020	XML AC: 1.1, 1.2, 1.3		<ul style="list-style-type: none"> <li>• Appendix C</li> </ul>

<b>11</b> 2: 21/09/2020 – 25/09/2020	Android Development AC: 1.1, 1.2, 1.3, 2.2, 2.3	Take Home Test 3	<ul style="list-style-type: none"> <li>• Appendix D</li> </ul>
<b>12</b> 2: 28/09/2020 – 02/10/2020	Android Development AC: 1.1, 1.2, 1.3, 2.2, 2.3		<ul style="list-style-type: none"> <li>• Appendix D</li> </ul>
<b>13</b> 2: 05/10/2020 – 09/10/2020	Android Development AC: 1.1, 1.2, 1.3, 2.2, 2.3	Project	<ul style="list-style-type: none"> <li>• Appendix D</li> </ul>
<b>14</b> 2: 12/10/2020 – 16/10/2020	<b>Revision</b>		<ul style="list-style-type: none"> <li>• All Chapters &amp; Appendix</li> </ul>
<b>15</b> 2: 19/10/2020 – 23/10/2020	<b>Take Home Assessment</b>		
<b>16</b> 2: 26/10/2020 – 30/10/2020	<b>Consultation and Take Home Assessment Feedback</b>		
2: 04/11/2020	<b>All Formative Marks Submitted</b>		

# Appendix A

This appendix should be referenced when studying the content and the assessment criteria for the week(s) listed below.

<b>Week</b>	1
<b>Learning Outcome</b>	LO1
<b>Assessment Criteria</b>	1.1, 1.2, 1.3

## Distributed Computing

### Supplementary Reading

Reilly, D (2002) *Java Network Programming and Distributed Computing*. ISBN-13: 978-0201710373

### Introduction

Distributed computing involves the design and implementation of applications as a set of cooperating software entities (processes, threads and objects) that are distributed across a network of machines.

### Distributed computing

Distributed computing can be defined as a computer processing technique, which allows different parts of an application to run simultaneously on two or more computers that are communicating with each other over a network.

The goal of distributed computing systems is to connect users and resources in a transparent, open and scalable manner. An example of distributed computing is Berkeley Open Infrastructure for Network Computing (BOINC), which is a framework in which large task problems are subdivided into smaller chunks of tasks and thereafter distributed to other computers for processing. The outcomes from each computer are reassembled into a larger solution.

Distributed computing is different from computer networking in that it is an interaction between computer nodes without the typical sharing of resources of a single program. Distributed computing makes use of computer networks to enable computer systems to communicate efficiently. Different technologies and standards are used to construct distributed computing, for example, Remote Procedure Call (RPC) and RMI.

For proper communication between computer nodes, the organisation of the interaction is of paramount importance. In such a scenario, the protocol or communication channel must not contain any information that may not be understood during the interaction.

### 1.1 Advantages of distributed computing

The advantages of distributed computing include:

- **Openness:**

The ability of a subsystem to continuously accept interaction with other systems; this allows system scalability.

- **Performance:**

Distributed computing involves communication and interaction between computers in a transparent and open manner; it, generally, increases performance over perfectly closed and self-contained systems.

- **Scalability:**

The ability to deploy re-usable distributed components on powerful servers.

- **Resource sharing:**

This allows many users access to a common database as well as the sharing of expensive devices.

- **Fault tolerance:**

Data may be replicated so that it exists in more than one site; the failure of a node or communication link does not necessarily make data inaccessible.

### 1.2 Difficulties with distributed computing

Difficulties with distributed computing include:

- Latency
- Synchronisation
- Partial failure

### 1.3 Distributed computing architectures

Various hardware and software architectures are used in distributed computing. Distributed programming falls under one of several basic architectures/categories, namely:

- **Client-server architecture:**

This involves the interaction between a client and server. Client code contacts a server by requesting data, upon which the server replies with a response. To commit a permanent change on the server, client input is relayed back to the server.

- **3-tier architecture:**

This architecture simplifies application deployment; it is mostly used by web applications. It moves client intelligence to the middle tier so that stateless clients can be used.

- **N-tier architecture:**

This is an extension of the 3-tier architecture. Applications that implement such further forward requests to other enterprise services.

- **Tightly coupled (clustered) architecture:**

This uses highly sophisticated machines to run the same process in parallel while subdividing each task into chunks and later reassembling such into a final solution.

- **Peer-to-peer architecture:**

This is an architecture wherein no particular machine provides services or manages resources on behalf of others. Responsibilities are thus uniformly distributed among the machines involved, known as 'peers'.

- **Space-based architecture:**

This is an architecture that creates the illusion (virtualisation) of a single address space. Data replication is transparent according to application needs; this efficiently manages time, space and reference.

Another aspect of distributed computing architecture worth mentioning is the method of communication and coordination in a concurrent process.

## **1.4 Distributed computing languages**

Nearly all programming languages that have access to the full hardware of a system could handle distributed programming given enough time and code. RPCs distribute operating system commands over network connections. Systems such as Common Object Request Broker Architecture (CORBA), Microsoft Distributed Component Object Model (DCOM), RMI and others try to map object-oriented design to networks. Loosely coupled systems communicate via intermediate documents that are, typically, human readable, e.g. Extensible Markup Language (XML), Hypertext Markup Language (HTML), etc. Designated languages for distributed programming include Ada, E, Erlang, Limbo, Oz and Orca.

## **Concluding remarks**

There are many different types of distributed computing system as well as many challenges to overcome when designing such. The main goal of distributed computing systems is to connect users and resources in a transparent, open and scalable manner. This creates a higher fault tolerance and more powerful systems than a combination of standalone computers.



# Appendix B

This appendix should be referenced when studying the content and the assessment criteria for the week(s) listed below.

<b>Week</b>	8 - 9
<b>Learning Outcome</b>	LO7
<b>Assessment Criteria</b>	7.1, 7.2, 7.3, 7.4

## Remote Method Invocation

### Prescribed reading

Liang, Y.D. 2018. *Introduction to Java programming and Data Structures*. 11th edition. Pearson. ISBN: 9781292221878 Chapter 40

### Introduction

RMI coordinates object function calls between Java Virtual Machines (JVMs) (JVMs can be located on different machines): a JVM can initiate methods belonging to another JVM. RMI allows applications to call object methods located remotely, thereby sharing resources and processing load across systems. Unlike other remote execution systems, RMI allows the use of any type of Java object.

### 1.0 RMI overview

Herewith a brief RMI overview:

- RMI provides distributed object capabilities for Java applications.
- RMI allows Java methods to obtain references to remote objects and invoke remote object methods nearly as easily as if such existed locally.
- RMI uses object serialisation to marshall and unmarshall method arguments.
- RMI supports the dynamic downloading of class files across networks.

## Case study

Consider the following scenario:

James creates a service that performs a useful function. He regularly performs function updates on said service, adding new features and improving existing ones.

Jane wishes to use the service provided by James, however, it is inconvenient and against company policy for James to supply Jane with an update every time. Jane uses RMI, since it supports the dynamic downloading of new classes, to handle updates automatically as James adds new features and optimises code execution. James only needs to place the new classes and updates in a web hierarchy for Jane to dynamically load such.

### 1.1 RMI stubs and skeletons

RMI uses stub and skeleton objects to provide connections between clients and remote objects.

A stub acts as a proxy for a remote object, which is responsible for forwarding method invocation from the client to the server where the actual remote object implementation resides. A client's reference to a remote object indirectly references to a local stub where the client has a local copy of said stub.

A skeleton, in turn, acts as a dispatcher for server-side objects, which contain methods that dispatch calls to actual remote object implementation. A skeleton object is automatically provided on the server's side. A remote object has an associate local skeleton to dispatch remote calls to.

A reference to a remote object via a method is:

- Using a directory service as a look-up service for the remote object; RMI uses what we call a 'RMI Registry' as the directory service
- Receiving the remote object reference as a method argument or return value

Figure 1 illustrates a RMI connection as made by a client. Using the case study, the client, Jane, initially contacts the RMI Registry and requests the name of the service. The exact location of James's RMI service is not known, however, Jane needs the location of his registry. The loading of updated classes happens transparently for Jane; no extra blocks of code are required to retrieve said classes. If the service changes often, Jane only needs to fetch the new subclasses from the web server where the two developers share classes:

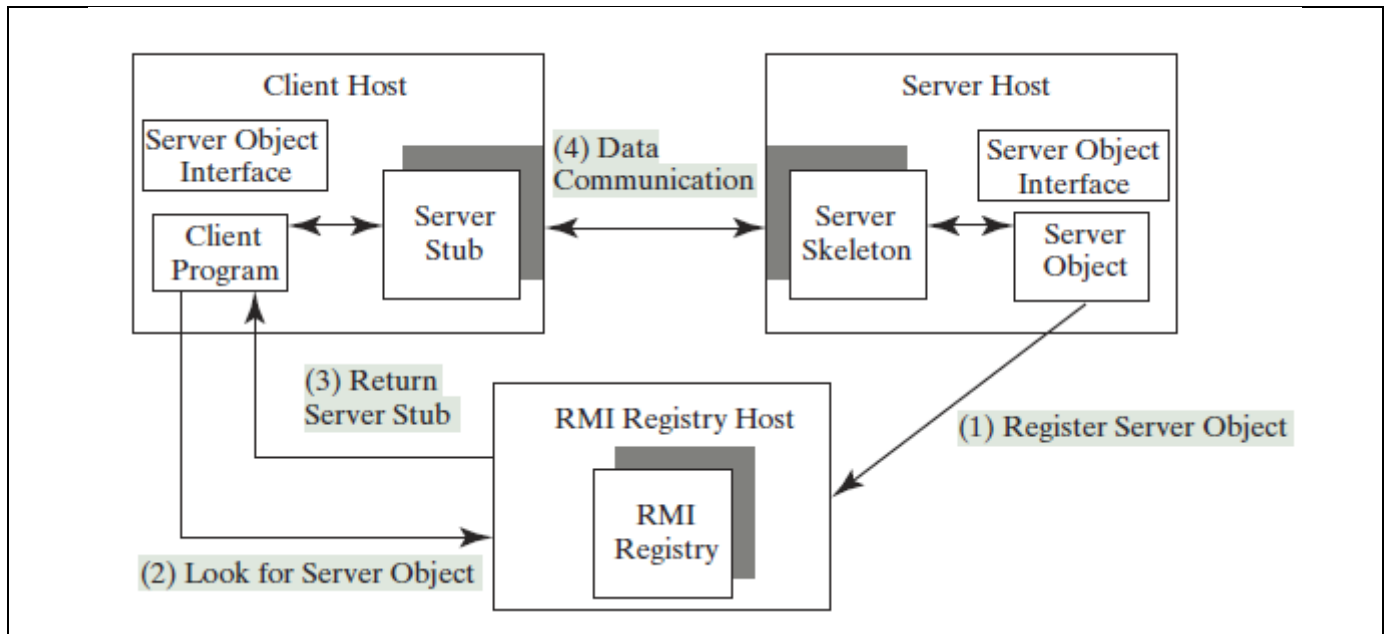


Figure 1 – RMI connection as made by a client

Source: Liang (2018)

## 1.2 Developing RMI applications

Writing your own RMI services can be a little difficult at first, so we will start off with an example that is not too ambitious. We will create a service that can calculate the square of a number and the power of two numbers (for example,  $2^{38}$ ). Owing to the large size of the numbers, we will use the `java.math.BigInteger` class for returning values rather than an integer or long.

Steps to develop a RMI application include the following:

- Design and implement the components of the distributed application:
  - Define the remote interface(s)
  - Implement the remote object(s)
  - Implement the client(s)
- Compile sources and generate stubs (and skeletons)
- Make the required classes network accessible
- Run the application

## 1.3 Defining interfaces

An interface is a method that contains abstract methods; such must be implemented by another class. An object becomes remote-enabled when implementing a remote interface, which has the following characteristics:

- A remote interface extends the `java.rmi.Remote Interface`.

- Each method of an interface should declare `java.rmi.RemoteException` in its throws clause in addition to any application-specific exceptions.

For the above-mentioned, let us consider the following:

- Creating a method that accepts integer as a parameter and returns a `BigInteger`: `public BigInteger square(int value);`
- Creating a method that accepts two integers as parameters, estimates their power and returns a `BigInteger`: `public BigInteger power(int x, int y);`

Figure 2 defines the desired remote interface:

```
import java.math.BigInteger; import java.rmi.*;

//Interface for a RMI service that calculates powers public interface
ISimpleService extends java.rmi.Remote {
//calculate the square of a number
public BigInteger square(int value) throws RemoteException;
//calculate the power of a number
public BigInteger power(int x, int y) throws RemoteException;
}
```

**Figure 2– How to define desire remote interfaces**

Source: Liang (2018)

A class that implements the above remote interface can be used as a remote object. Clients can remotely invoke the `square(int value)` method, which will return the square of the number in parameter to the client.

## 1.4 Implementing interfaces

This section includes the constructor and main method. A declared default constructor does not necessarily have an initialisation code for our service. The fundamental reason why we need a default constructor is to be able to handle exceptions thrown from the parent constructor in `UnicastRemoteObject`.

Our default constructor throws `java.rmi.RemoteException` (Figure 3):

```
public SimpleServiceServer() throws RemoteException { super();
}
```

**Figure 3 – Default constructor**

## 1.5 Implementing servers

A Server class has a main method that creates an instance of the remote object implementation as well as exports and binds such an instance to a name in the Java RMI Registry.

The key functional responsibilities of a main method are:

- Creating an instance of the implemented interface while registering the remote object with the RMI Registry.
- Creating and exporting the remote object.
- Assigning a SecurityManager to the JVM in order to prevent untrusted clients from using the service.

Figure 4 illustrates how to implement Server:

```
public static void main(String[] args) throws Exception {
    //assign a security manager in the event that dynamic classes are
    loaded if (System.getSecurityManager() == null)
        System.setSecurityManager(new RMISecurityManager());
    //create an instance of our power service server...
    SimpleServiceServersvr = new SimpleServiceServer();
    //...and bind it with the RMI Registry
    Naming.bind ("SimpleService", svr);
    System.out.println ("Service bound...");
}
```

Figure 4 – How to implement Server

### Note

A class can define methods that are not declared in the remote interface, however, only the virtual machine running the service can invoke such.

Figure 5 illustrates the full source code for a RMI Server:

```
import java.math.*;
import java.rmi.*;
import java.rmi.server.*;

//SimpleServiceServer
//Server for a RMI service that calculates power
```

```

public class SimpleServiceServer implements ISimpleService {
    public SimpleServiceServer() throws RemoteException {
        super();
    }

    //calculate the square of a number
    public BigInteger square(int number) throws RemoteException {
        String numrep = String.valueOf(number);
        BigInteger bi = new BigInteger (numrep);
        //square the number
        bi.multiply(bi);
        return(bi);
    }

    //calculate the power of a number
    public BigInteger power(int num1, int num2) throws RemoteException {
        String numrep = String.valueOf(num1);
        BigInteger bi = new BigInteger (numrep);
        bi = bi.pow(num2);
        return bi;
    }

    public static void main (String args[]) throws Exception {
        //assign a security manager in the event that dynamic classes are
        loaded if (System.getSecurityManager() == null)
        System.setSecurityManager (new RMISecurityManager());
        try {
            //create an instance of our simple service server...
            SimpleServiceServersvr = new SimpleServiceServer();
            SimpleService service = (SimpleService) UnicastRemoteObject.ex
portObject (svr, 0);
            //... and bind it with the RMI
            Registry Registry = LocateRegistry.getRegistry();
            registry.bind ("SimpleService", service);
            System.out.println ("Service bound...");
        }
    }
}

```

```
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}  
}
```

**Figure 5 – RMI Server**

## 1.6 Creating and exporting remote objects

The previous code showed that the main method creates a remote object and exports such to the Java RMI runtime to receive incoming remote calls. The code scope that exports the supplied remote object to receive incoming calls on a TCP port and returns stub for the remote object to pass to clients is represented in Figure 6:

```
SimpleServiceServersvr = new SimpleServiceServer();  
SimpleService service = (SimpleService)UnicastRemoteObject.exportObject  
t (svr, 0);
```

**Figure 6 – How to create and export remote objects**

With the help of `exportObject`, runtime begins to listen to new server socket or, alternatively, uses a shared server socket to accept incoming calls for the remote object.

## 1.7 Registering remote objects with a RMI Registry

For initialisation, Java RMI provides a registry API for customised applications to bind names to remote object stubs, while also allowing clients to look up remote objects by name in order to obtain their reference (stub).

Figure 7 obtains a reference for a registry on the local host while using the default registry port:

```
Registry registry = LocateRegistry.getRegistry();  
registry.bind ("SimpleService", service);
```

**Figure 7 – How to register remote objects with a RMI Registry**

The `getRegistry` method of the `Registry` class returns a stub that implements the remote interface `java.rmi.registry.Registry`. This method uses the default registry port, **1099**, to send invocation to the registry on the server's local machine. The `bind` method is responsible for binding the remote object's reference to the name `SimpleService` in the registry.

## 1.8 Implementing clients

To implement a client program, you must:

- Assign a **SecurityManager**.
- Obtain a stub for the registry on the machine from where the server runs.
- Search the remote object's reference (stub) by name in the registry.
- Invoke the power and square methods on the remote object using the stub.

Figure 8 illustrates an example of such:

```
import java.rmi.*;
import java.rmi.Naming;
import java.io.*;

//SimpleServiceClient

public class SimpleServiceClient {

public static void main(String args[]) throws Exception {
    //check for hostname argument
    if (args.length != 1) {
        System.out.println("Syntax - SimpleServiceClient host"); System
m.exit(1);
    }

    //assign security manager
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }

    //call registry for PowerService
    SimpleService service = (PowerService) Naming.lookup ("rmi://" + a
rgs[0] + "/SimpleService");
    DataInputStream din = new DataInputStream (System.in);

    for (;;) {
```



```

System.out.println("1 - Calculate square");
System.out.println("2 - Calculate power");
System.out.println("3 - Exit");
System.out.println();
System.out.print("Choice : ");

String line = din.readLine(); Integer choice = new Integer(line);

int value = choice.intValue();

switch (value) {
    case 1:
        System.out.print ("Number : "); line = din.readLine();

        System.out.println();
        choice = new Integer (line); value = choice.intValue();

        //call remote method
        System.out.println("Answer : " + service.square(value));

        break;

    case 2:
        System.out.print ("Number : ");
        line = din.readLine();
        choice = new Integer (line);
        value = choice.intValue();
        System.out.print ("Power : ");
        line = din.readLine();
        choice = new Integer (line);
        int power = choice.intValue();
        //call remote method
        System.out.println("Answer : " + service.power(value,
power));

        break;

```

```

        case 3:
            System.exit(0);

        default:
            System.out.println ("Invalid option");
            break;
    }
}
}
}
}

```

**Figure 8 – How to implement client service programs**

To identify a stub for the registry on a server's host, we specify a RMI URL. This URL consists of a logical name for the stub and hostname machine where the service is located. The instance object returned could now be used to call remote methods as if the class was declared locally. The code snippet that represents a call registry for `SimpleService` is represented in Figure 9:

```

SimpleService service = (PowerService) Naming.lookup ("rmi://" + args[
0] + "/SimpleService");

```

**Figure 9 – How to identify a stub for the registry on the server's host**

## 1.9 Running servers and clients

We can now compile the client and server codes:

- `javac ISimpleService.java`
- `javac SimpleServiceServer.java`
- `javac SimpleServiceClient.java`

Use the `rmic` utility to generate the required Reference(stub) and Skeleton classes: `Rmic SimpleServiceServer`

The above command generates the Stub and Skeleton classes:

`SimpleServiceServer_Stub.class`

and

`SimpleServiceServer_Skel.class` (not needed in Java 2)

The next step would be to make the class files network accessible. For the moment, let us assume that all class files are available locally to both the client and server via their CLASSPATH. That way, we do not have to worry about dynamic class downloading over the network. We will now see how to properly handle such a situation:

Each client must have the following files in its CLASSPATH:

- `ISimpleService.class`
- `SimpleServiceClient.class`
- `SimpleServiceServer_Stub.class`

The server program must have the following files in its CLASSPATH:

- `ISimpleService.class`
- `SimpleServiceClient.class`
- `SimpleServiceServer_Stub.class`
- `SimpleServiceServer_Skel.class`

To run the application in Java 2, a security policy file is needed that will allow the downloading of class files. Figure 10 is an example of a policy file that allows anything:

```
grant {  
  permission java.security.AllPermission;  
};
```

**Figure 10 – How to create security policies to run RMI applications**

A policy that allows a program to connect to/accept connections from any host on ports greater than 2000 and to connect to any host on Port 80 would be as follows (Figure 11):

```
grant {  
  permission java.net.SocketPermission "*:1024-65535", "connect, accept";  
  permission java.net.SocketPermission "*:80", "connect";  
};
```

**Figure 11 – How to create security policies to run RMI applications**

## 1.10 Summary

The Server code discussed implements a generic compute engine. The basic idea of RMI is for a client application to encapsulate tasks to be done as objects and to send such remotely to a server for execution. The server's computational power is used to run said tasks and return the results to the client.

### **Concluding remarks**

Java API is used in distributed applications, which allow the distribution of objects; such allows a Java method to obtain a reference to a remote object and invoke the methods of such nearly as easily as if the remote object existed locally. This appendix discussed the ways in which to implement RMI clients and servers, export remote objects and register remote objects with a RMI Registry.

# Appendix C

This appendix should be referenced when studying the content and the assessment criteria for the week(s) listed below.

Week	10
Learning Outcome	LO1
Assessment Criteria	1.1, 1.2, 1.3

## eXtensible Markup Language (XML)

### Supplementary reading

- Morrison, M. 2006. *Sams teach yourself XML in 24 hours*. Indianapolis: Sams.
- Steele, J. & To, N. 2011. *The Android developer's cookbook: building applications with the Android SDK*. New Jersey: Addison-Wesley.

### Introduction

XML is used to describe data and create tags. XML was created by the World Wide Web Consortium (W3C) to overcome the limitations of HTML, which forms the basis for all web pages.

The XML standard is a flexible way of creating information formats and electronically sharing structured data via the public Internet as well as via corporate networks. It is widely used for the definition of device- and system-independent methods used to store and process texts in electronic form. It is now also the interchange and communication format used by many applications on the World Wide Web (WWW).

XML is a metalanguage, that is, a language that is used to describe other languages; in this case, markup languages. XML has **three** highly distinctive advantages, namely:

1. It places emphasis on descriptive rather than procedural markup.
2. It distinguishes between syntactic correctness and validity (DTDs).
3. It is independent of any **one** hardware or software system; applications can use XML to store, transmit and display data.

### 1.1 XML syntax

To be able to create and edit XML documents, you need an editor. For the purposes of

this module, we will employ Notepad as such.

An XML document contains a prologue and body. The prologue consists of an XML declaration, possibly followed by a document type declaration. The body is made up of a single root element, possibly with some comments and/or processing instructions. An XML document is, typically, a computer file whose content meets the requirements of XML specification.

## 1.2 XML declaration

An XML declaration is the first few characters of an XML document; such is also known as the 'processing instruction'. The afore-mentioned is a line of code at the beginning of an XML document that identifies the XML version used by said document. Such a declaration is used by processing software to work out how to deal with the subsequent XML content.

A typical XML declaration is shown below. The encoding of documents is particularly important as XML processors will default to UTF-8 when reading an 8-bit-per-character document:

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

## 1.3 Elements and attributes

Data is divided into elements and attributes. Each element is surrounded by a start tag and end tag. The segment of an XML document between an opening and corresponding closing tag is called an 'element'. Elements may contain text or other elements. An end tag resembles its start tag but includes a slash before the closing tag name, for example:

```
<tel>0114501963</tel>
```

An XML file consists of elements: each has a tag name and is surrounded by start and end tags. Tag names should be chosen to describe the data, for example:

```
<name>Michael Mapundu</name>
```

Elements can contain data or sub-elements. A **person** element, for example, would contain **title**, **name**, **tel** and **email** sub-elements. Elements can contain attributes, which provide additional information, for example:

```
<person ssn = "123 4589">
```

Figure 12 illustrates an example of a simple XML document:

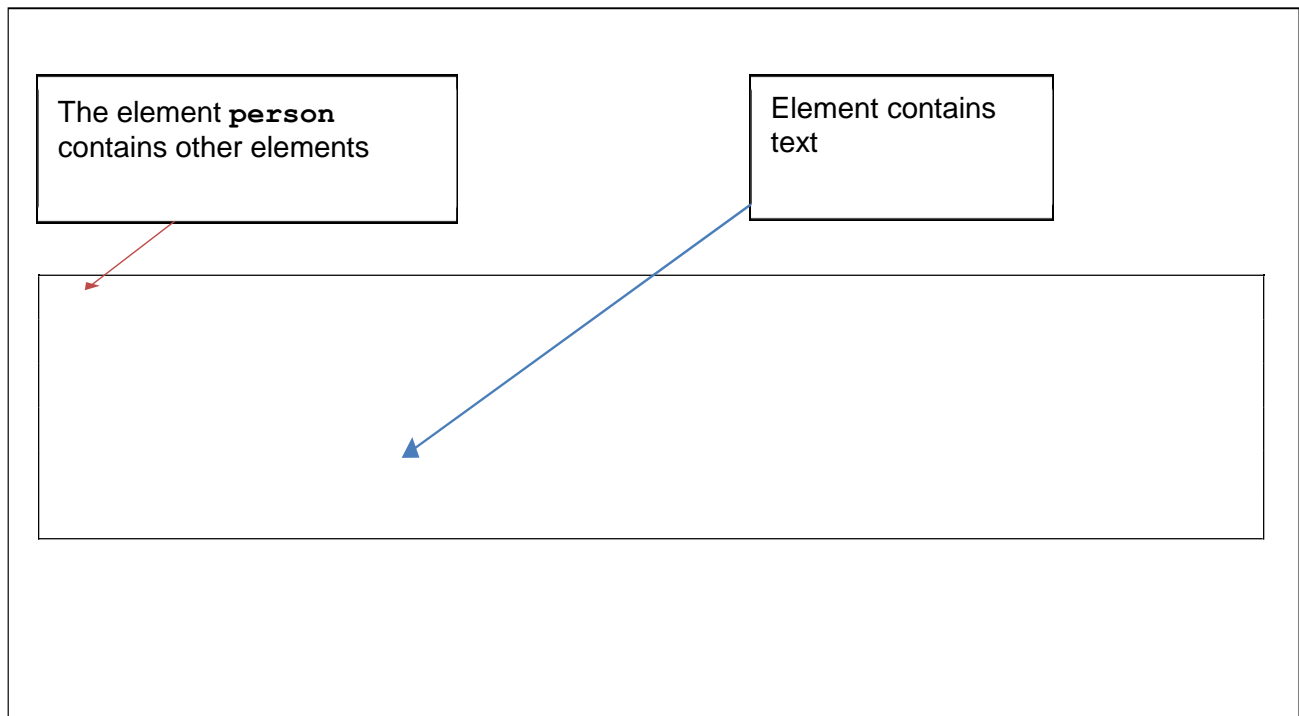


Figure 12 – Simple XML documents

Source: Michael Mapundu (2015)

An opening tag may contain attributes. These are, typically, used to describe element content (Figure 13):

```
<entry>
  <word language = "en">cheese</word>
  <word language = "fr">fromage</word>
  <word language = "ro">branza</word>
  <meaning>A food made ...</meaning>
</entry>
```

Figure 13– How to declare attributes within opening tags

Using an attribute rather than an element might make the structure more difficult to alter in future:

- Multiple values are **not** permitted.
- Tree structures are **not** permitted.

A general rule is to avoid using an attribute unless there is a good reason for doing so:

- Use an attribute to describe how data should be interpreted (e.g. language, currency, etc.).
- Use an attribute for 'identities' (IDs) (e.g. identifying data).

## 1.4 XML documents as trees

XML documents are abstractly modelled as trees as reflected by their nesting. Sometimes, XML documents are graphs.

Figure 14 illustrates an example of an XML document as a tree:

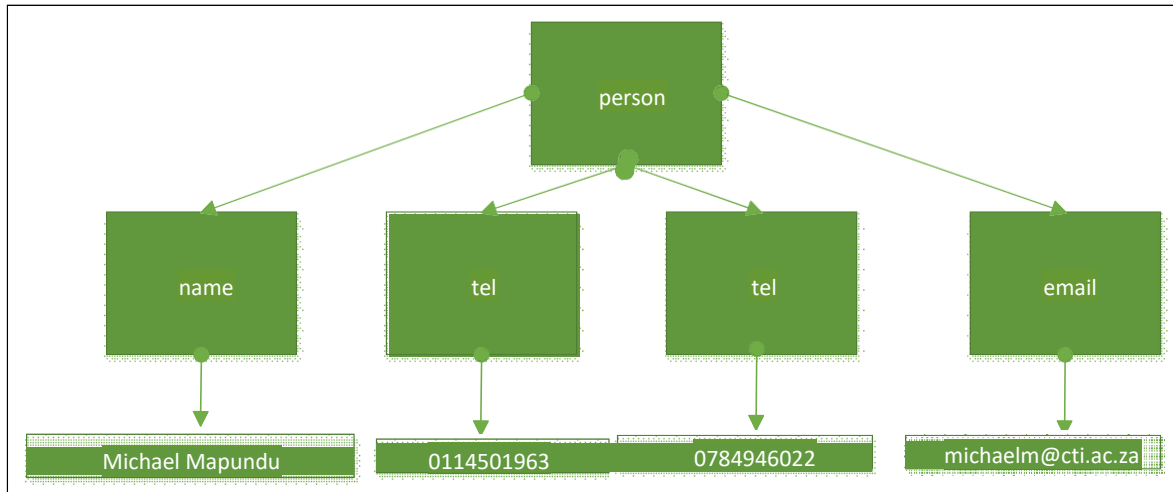


Figure 14 – XML documents as trees

Source: Michael Mapundu (2015)

Figure 15 illustrates an example of a complete XML document:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<addresses>
  <person ssno = "113">
    <name>Michael Mapundu</name>
    <tel>0114501963</tel>
    <tel>0784946022</tel>
    <email>michaelm@cti.ac.za</email>
  </person>
```

Figure 15 – Complete XML documents

Source: Michael Mapundu (2015)

## 1.5 Viewing XML documents

Style sheets are employed to view XML documents. These allow you to carefully control and manage web page content in your default web browser. This implies that colour and font, thus how you want documents to appear, can be controlled.

Figure 16 illustrates an example of a style sheet:



```

tt {
    display: block;
    width: 750px;
    padding: 10px;
    margin-bottom: 10px;
    border: 4px double black;
    background-color: silver;
}

question {
    display: block;
    color: black;
    font-family: Times,
    serif; font-size: 16pt;
    text-align: left;
}

```

Figure 16 – Style sheets

## 1.6 XML rules

A few XML rules to keep in mind are:

- XML is order-sensitive.
- Tags come in pairs thus every opening tag must have a corresponding closing tag.
- XML is case-sensitive, i.e. tag names.
- Elements must be properly nested.
- There is a special shortcut for tags that have no text or sub-elements in between them (empty element, bachelor tag).
- There should be exactly **one** top-level element (root element).

## 1.7 Well-formed XML documents

For XML documents to be well formed, they should have the following:

- **One** top-level element
- Tags in properly nested, case-sensitive pairs
- Empty elements may use the accepted shortcut /
- Attribute values must be enclosed in quotes
- Attribute names must **not** be repeated within tags

## Note

If a well-formed XML document is loaded by your web browser, it will display as a tree structure. This is perhaps the simplest way to check whether such is well formed. If **not**, the XML that is loaded will display error messages.

### 1.8 XML IDs and referencing

Unique elements can be identified with IDs and referred to and from other elements. In this way, element relationships can be shown without repetition.

Figure 17 illustrates an example wherein each person has an ID and can contain a reference to the ID of their mother, father and children respectively:

```
<family>
<person id = "lisa" mother = "marge" father = "homer">
<name>Lisa Simpson</name>
</person>

<person id = "bart" mother = "marge" father = "homer">
<name>Bart Simpson</name>
</person>

<person id = "marge" children = "bart lisa">
<name>Marge Simpson</name>
</person>

<person id = "homer" children = "bart lisa">
<name>Homer Simpson</name>
```

Figure 17– How to implement IDs

### 1.9 XML schemas

XML schemas are used to define the structure and data of XML documents. The entire process is also known as ‘data modelling’ as it entails resolving classes of data into elements and attributes. XML documents should thus conform to schemas for such to be valid; conform to regular-expression grammar; and ensure that the types of attribute are correct and constraints on references satisfied. Moreover, the afore-mentioned will enable developers to check if XML

documents adhere to the sets of constraints as defined in schemas. This, further, implies that documents obey schema rules. Such is important as it allows XML documents to be validated for accuracy and consistency (Morrison, 2006).

Figure 18 illustrates an example of a simple XML document with constraints:

```
<?xml version = "1.0" encoding = "UTF-8"?> processing instruction
<people> top-level element
    <person ssn = "3245"> element containing sub-
elements and an attribute
    <title>Mr</title> optional element
    <name>Michael Mapundu</name> required element
    <tel>0114501963</tel> at least one tel number
    <tel>0784940838</tel>
    <email>michaelm@cti.ac.za</email> up to two emails are optional
    </person>
...any number of person elements
</people>
```

Figure 18 – Simple XML documents with constraints

Source: Michael Mapundu (2015)

## 1.10 DTDs

**Document Type Definitions** (DTD) allow you to check that well-formed files contain the correct elements in the correct quantities. You, therefore, need to write specifications for XML files. DTDs are original and relatively simple to implement. In such, you can specify the permitted content for each element using regular expressions. A regular expression describes the 'pattern' of a string in a concise and flexible manner. Regular expressions are very powerful. This module will only use simple examples.

In Figure 18, we can add the following constraints:

For the **person** element, the regular expression is:

name, title?, tel+, email\* This implies that:

- **name** = there must be a **name** element.
- **title?** = there is an optional **title** element (i.e. **zero** or **one** title elements).
- **name, title?** = the **name** element is followed by the optional **title** element.
- **tel+** = there are **one** or more tel elements.

- **email\*** = there are **zero** or more e-mail elements.

Figure 19 illustrates an example of a DTD:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE people [
<!ELEMENT people (person+)>
<!ELEMENT person
(name, title?, tel+, email*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT tel (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ATTLISTperson
ssn CDATA REQUIRED> ]>
```

Figure 19 – DTDs

Source: Michael Mapundu (2015)

Interpreting Figure 19 can be done as follows:

- The top-level element is **people**.
- The **people** element consists of **one** or more **person** elements.
- The **person** element consists of **name**, **title**, **tel** and **email**, with the previously mentioned constraints.
- The **name**, **title**, **tel** and **email** elements each consist of character data.
- The **person** element has a required attribute consisting of character data (**ssn**).

### 1.10.1 Linking XML documents to DTDs

A DTD can be internal (part of the document file) (Figure 20):

```
<?xml version = "1.0"?>
<!DOCTYPE db [<!ELEMENT... > ...]>
<db>...</db>
```

Figure 20 – How to declare internal DTDs

Source: Michael Mapundu (2015)

A DTD can be external (the DTD and document are in different files). The following code illustrates an example of a DTD from a local file system:

```
<!DOCTYPE db SYSTEM "schema.dtd">
```

The following code illustrates an example of a DTD from a remote file system:

```
<!DOCTYPE db SYSTEM "http://www.schemaauthority.com/schema.dtd">
```

DTDs are rather weak specifications when going according to database and programming language standards. Their drawbacks include:

DTDs only have **one** base type (**PCDATA**).

There is no easy way of specifying constraints, e.g. range of values, frequency of occurrence, etc.

They are **not** easily parsed since they are **not** XML.

As a result of the above, DTDs are now being superseded by XML schemas.

### 1.11 XML schema

XML schemas are more precise and, therefore, complicated than DTDs. This implies that they are more powerful, flexible and intuitive than DTDs. Schemas were designed to replace DTDs but DTDs are very well established and simpler to use. The latter are written using XML syntax thus they can be parsed and validated with standard XML tools. XML schema-specific tags are used, such as `xs:element` to describe an element in a document; `xs:` is a prefix associated with a namespace. Furthermore, they have data types other than `#PCDATA` as well as some basic built-in simple types, such as `xs:string`, `xs:decimal`, `xs:integer` and `xs:ID` (you can also define your own types).

XML schemas allow developers greater control over permitted constructs thus maximum and minimum occurrences of each element can be easily specified (the default value of **minOccurs** and **maxOccurs** is 1); a set of permitted values can be specified, regular expressions can be used to set patterns to be matched, modularity and inheritance is supported and types can be named and referred to.

Let us have a look at the person element as an example (Figure 21):

```
<xs:element name = "person" maxOccurs = "unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "title" type = "xs:string" minOccurs = "0"/>
      <xs:element name = "name" type = "xs:string"/>
      <xs:element name = "tel" type = "xs:string"/>
    
```

```

<xs:element name = "email" type = "xs:string" minOccurs = "0" maxOccurs
s = "2"/>
</xs:sequence>
<xs:attribute name = "ssn" type = "xs:positiveInteger" use = "required
"/>
</xs:complexType>
</xs:element>

```

Figure 21 – XML schemas

Interpreting Figure 21 can be done as follows:

- The **title** element is specified as **<xs:element name = "title" type = "xs:string" minOccurs = "0"/>**.
- The **xs:element** tag is used to specify elements.
- The tag **name** is specified using the **name** attribute.
- **type** is specified using the **type** attribute; the built-in XML schema type for strings is **xs:string**.
- The maximum occurrence is **not** specified thus the default value **1** will be used.
- **xs:element** is an empty element (i.e. it has no data or sub-elements) thus it is closed with a slash instead of an end tag.
- Elements can be simple as is the case with **title**.
- Elements can be more complex as is the case with **person**.
- The **person** element has a complex type; such consists of:
  - A sequence of other elements (**title**, **name**, **tel** and **email**)
  - The required attribute **ssn**

### Note

For optional attributes, omit **use = "required"**.

- The first **two** lines in Figure 108 are standard.
- The top-level element is **people**; it is a complex type consisting of a sequence of **person** elements.
- There will be at least **one person** element (default value of **minOccurs**); there is no upper limit to the number of **person** elements.
- Save a schema such as this in a file with extension **xsd**.
- Linking schema definitions with the XML document is done via the special attribute
- **xmlns:xsi** of the root node of the document (Figure 22):

```
<people xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation = "addresses1.xsd">
```

**Figure 22 – How to link XML schemas with special attributes**

In Figure 22, the specification file, `addresses1.xsd`, is in the same directory as the XML file. The attribute `xsi:noNamespaceSchemaLocation` is used because the XML file does not have its own namespace.

## 1.12 XML namespaces

Namespaces provide a way of establishing unique identifiers and elements. It is important that each and every namespace has a unique name. Namespaces have to be tied to uniform resource identifiers (URIs). URIs are, usually, used to reference physical resources on the Internet and thus have to be unique.

In a nutshell, a namespace is the name of a unique URI. Namespaces apply to whole XML documents. There are namespaces associated with used tags that let such be used unambiguously. A namespace is identified by a short prefix, e.g. `xs`, and a unique URI, for example:

```
xmlns:Prefix = "Namespace"
```

The **xmlns** attribute is what informs an XML processor that a namespace is being declared. The namespace part identifies the URI that is unique to the elements and attributes. The **prefix** serves as a shorthand reference to the namespace throughout the scope of the element in which it is declared.

## 1.13 Declaring and using namespaces

You have to declare a namespace for a particular element with the scope that you want for such. When you declare a namespace, it looks like an attribute of the element.

At the start of a document, you must specify what namespaces are being used. Default declaration is when a namespace is declared without a prefix thus all elements and attribute names within the scope are referenced, using unqualified names, and assumed to be in said namespace. Unqualified namespaces do **not** include a prefix and are, usually, associated with a default namespace or nothing at all. Qualified namespaces include a prefix as well as the local portion of the name. Explicit declaration is when a namespace is declared with a prefix thus all elements and attributes are associated with such.

Figure 23 illustrates an example of declaring a namespace in an XML schema:

```
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema"
  xmlns = "http://example.org/pp"
  targetNamespace = "http://example.org/pp"
  elementFormDefault = "qualified" attributeFormDefault = "qualified"
">
<xs:element name = "people">
...etc
```

Figure 23 – How to declare namespaces in XML schemas

Figure 24 illustrates an example of declaring a namespace in an XML file:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<pp:people
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-
instance" xmlns = "http://example.org/pp"
  xsi:schemaLocation "http://example.org/pp addresses1.xsd">
<pp:person ssn = "4444">
<pp:title>Mr</pp:title>
...etc
```

Figure 24 – How to declare namespaces in XML files

## 1.14 XML data constraints

You can constrain data values in:

- Range (Figure 25):

```
<xs:minInclusive value = "0"/>
<xs:maxInclusive value = "120"/>
```

Figure 25 – Data values range constraint

- Length (also **minLength** and **maxLength**):

```
<length value = "3"/>
```

- Pattern:

```
<xs:pattern value = "([a-z])*"/>
```

The above means **0** or more lowercase alphabetic **chars**.



- Enumerated list (Figure 26):

```
<xs:enumeration value = "Audi"/>
<xs:enumeration value = "Golf"/>
<xs:enumeration value = "BMW' />
```

Figure 26 – Data values enumerated list constraint

An attribute of a named type:

- A named type is declared (Figure 27):

```
<xs:simpleType name = "ssstype">
<xs:restriction base = "xs:positiveInteger">
<xs:maxInclusive value = "999999"/>
</xs:restriction>
</xs:simpleType>
```

Figure 27 – How to declare name types

- The above is then used as an attribute type:

```
<xs:attribute name = "ssn" type = "ssstype" use = "required"/>
```

### 1.15 XML keys and keyrefs

In XML schemas, keys and keyrefs can be defined instead of IDs and IDrefs. Keys are any unique, non-null elements or attributes. Keyrefs are elements which contain the value of said keys; they can be used to refer to such. For example, each holiday home has people who are contacts; contact details are stored separately to the holiday homes to avoid duplication (Figure 28):

```
<homelist>
<homes>
<home id = "1">
    <hname>Rose Cottage</hname>
    <location>Inverness</location>
    <cID>C1</cID> keyref
    <cID>C2</cID>
</home>
...more homes
</homes>
```

```

<contacts>
<contactdetails cID = "C1"> key
    <cname>John Smith</cname>
    <phone>01311231234</phone>
</contactdetails>
...more contact details
</contacts>
</homelist>

```

Figure 28 – How to declare keys and keyrefs

The procedure works as follows:

- Give the key a name:  
contactKey
- Use a selector to define which element has a key:  
contactdetails in contacts
- Use a field to define which element or attribute is the key: the **cID** attribute (Figure 29):

```

<xs:key name = "contactKey">
    <xs:selector xpath = "homelist/contacts/contactdetails"/>
    <xs:field xpath = "@cID"/>
</xs:key>

```

Figure 29 – How to define keys

- Give the keyref a name:  
contactRef
- Specify which key it refers to:  
contactKey
- Use a selector to define which element contains the reference:  
home
- Use a field to define which element or attribute is the reference: the **cID** element (Figure 30):

```

<xs:keyref name = "contactRef" refer = "contactKey">
    <xs:selector xpath = "homelist/homes/home"/>
    <xs:field xpath = "cID"/>

```

```
</xs:keyref>
```

Figure 30 – How to define references

## 1.16 XPath

Data stored in XML documents must be extracted to use such with various applications. XPath is a declarative language for extracting data from XML and is used extensively in other specifications, e.g. XQuery, XSLT and extended SQL. XPath provides an abstract means of addressing XML document parts and forms the basis of addressing XSLT. XPath is used to define parts of XML documents and perform simple calculations. Constraints can be applied to limit the parts extracted. The purpose of XSLT is to process XML document nodes and apply a technique of pattern matching and transformation. Moreover, XSLT style sheets consist of **one** or more templates that specifically describe patterns and expressions, which are used to match XML content with transformation purposes. The basic constructs will now be discussed.

## 1.17 Templates

A template is an XML construct that describes the output to be generated based on predefined pattern-matching criteria. Such is applied to nodes or groups of elements, for example (Figure 31):

```
<xsl:template match = "/">
    ...
</xsl:template/>
```

Figure 31– How to apply template matches

The / indicates that you are applying a match to the root element thus you end up applying such to the whole document. Note the following:

- **xsl:value-of:**  
Such inserts the value of an element or attribute.
- **xsl:if:**  
Such performs a conditional selection.
- **xsl:for-each:**  
Such loops through elements in a document.
- **xsl:apply-templates:**  
Such applies a template in a style sheet.

## Note

**xsl:value-of** requires an attribute select that identifies the specific content to be inserted.

### 1.18 Patterns and expressions

Patterns and templates are used in XSLT templates to perform matches; they determine the portions of XML documents that are passed through particular templates for transformation. Patterns describe the hierarchical nature of trees and their nodes. Expressions determine the parts of nodes selected for transformation. There are two types of expression, namely:

- 1 Returning a set of nodes (node sets)
- 2 Returning a value (**number**, **string**, **Boolean**)

To get text, you can use the node type text function `/catalog/cd/title/text()`. This will return all title text nodes. It is not necessary to use the text function in conditions.

### 1.19 Aggregation functions

The aggregation functions **count** and **sum** are applied to node sets and return numbers:

- A **count** result is always a number, e.g. `count(//artist)`.
- A **sum** result is only a number if every node in an argument set can be cast as a number, e.g. `sum(//cd[price>10]/price)`.
- Functions **min**, **max** and **avg** are new in XPath 2.0.

### 1.20 Expression interpretation

“/” has several meanings, namely:

- At the beginning of an XPath expression, “/” represents the root node of the document; such contains the root element.
- “/” between element names represents a parent-child relationship.
- “//” represents an ancestor-descendent relationship.
- It can be used as a shortcut to ‘all titles’: **//title**.

Rather than **/catalog/cd/title**:

- **@** marks an attribute
- **[condition]** specifies a condition, e.g. **[price = 10]**
- 

When using XPath, XML documents are considered to be trees with several types of node,

such as:

- **Element nodes:**

Such contain everything from the start to end tags. This may be data, sub-elements or attribute nodes, e.g. **<title>Dark side of the moon</title>**.

- **Attribute nodes:**

Such contain complete attributes, e.g. **country = "UK"**.

- **Text nodes:**

Such contain element or attribute text, e.g. **"UK"** or **"David Bowie"**.

Figure 32 illustrates an example of such:

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<catalog>
  <cd country = "UK">
    <title>Dark side of the moon</title>
    <artist>Pink Floyd</artist>
    <price>10.90</price>
  </cd>

  <cd country = "UK">
    <title>Space oddity</title>
    <artist>David Bowie</artist>
    <price>9.90</price>
  </cd>

  <cd country = "USA">
    <title>Aretha: Lady Soul</title>
    <artist>Aretha Franklin</artist>
    <price>9.90</price>
  </cd>
</catalog>
```

**Figure 32 – Hierarchical XML documents**

Figure 33 illustrates an example of a hierarchical XML document structure:

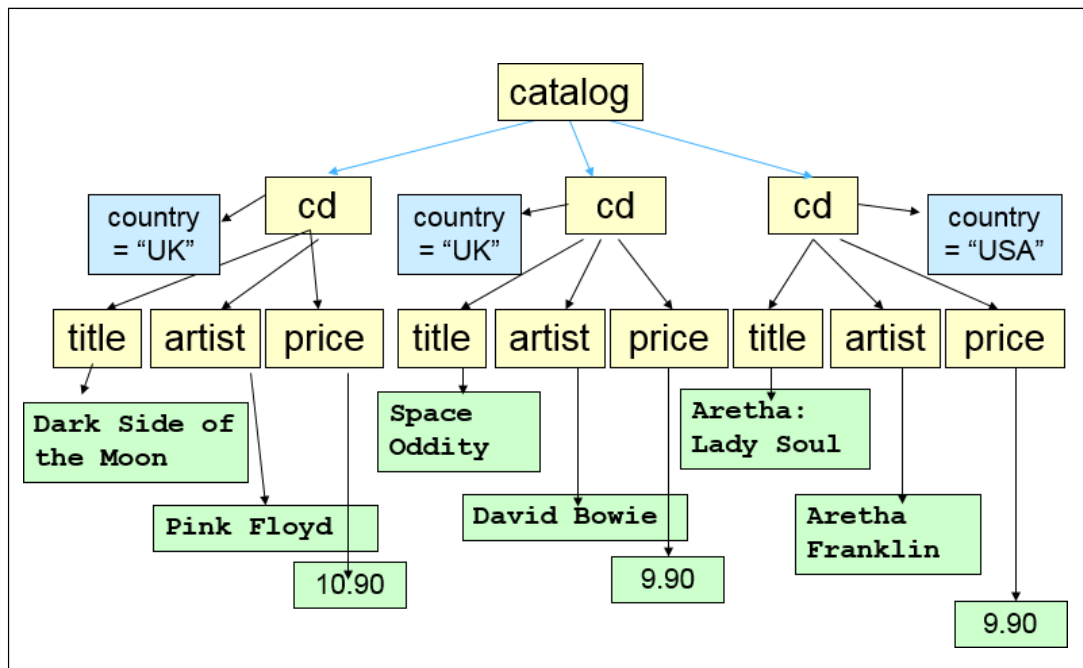


Figure 33 – Hierarchical XML document structures

Source: King (2014)

Figure 34 illustrates an example of extracting prices:

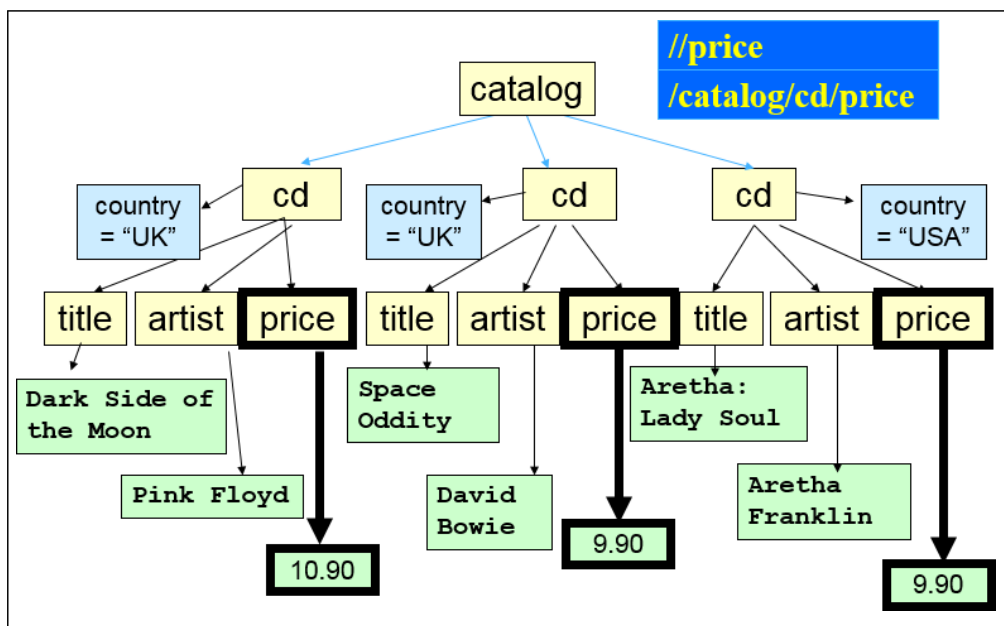


Figure 34 –How to extract prices

Source: King (2014)

Figure 35 illustrates an example of extracting prices using the text function:

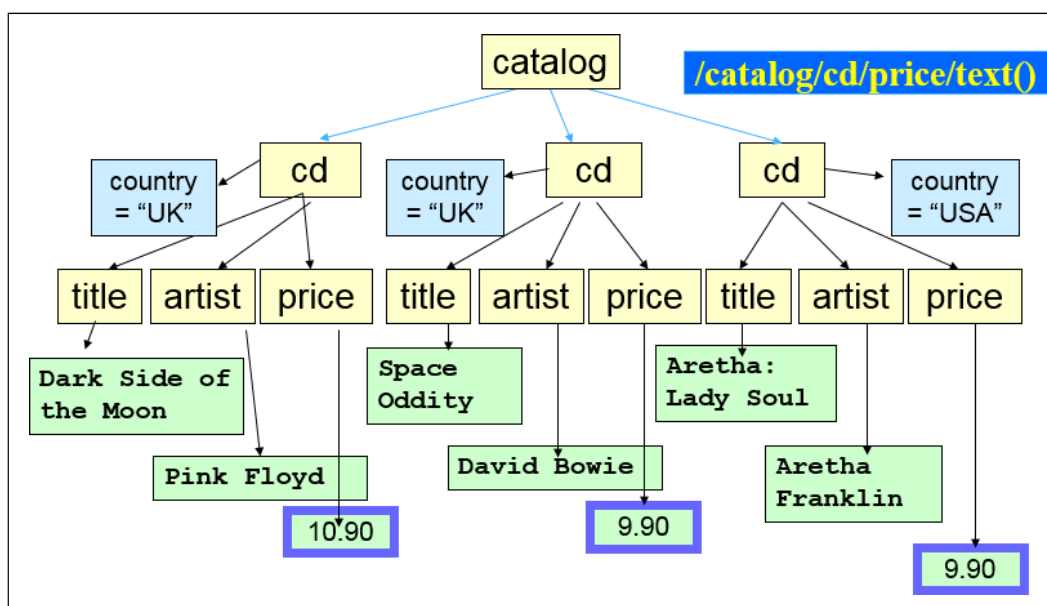


Figure 35 – How to extract prices using the text function

Source: King (2014)

Figure 36 illustrates an example of extracting prices with a condition of price of less than 10:

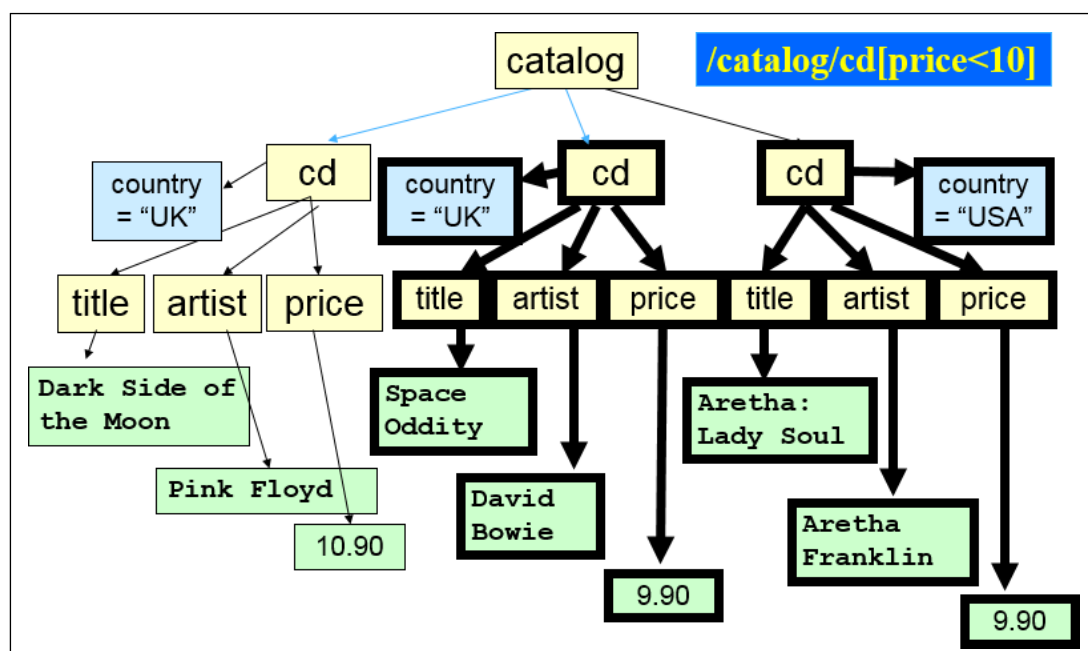


Figure 36 – How to extract prices with conditions of price of < 10

Source: King (2014)

Figure 37 illustrates an example of using an attribute within a condition to extract price using the text function:

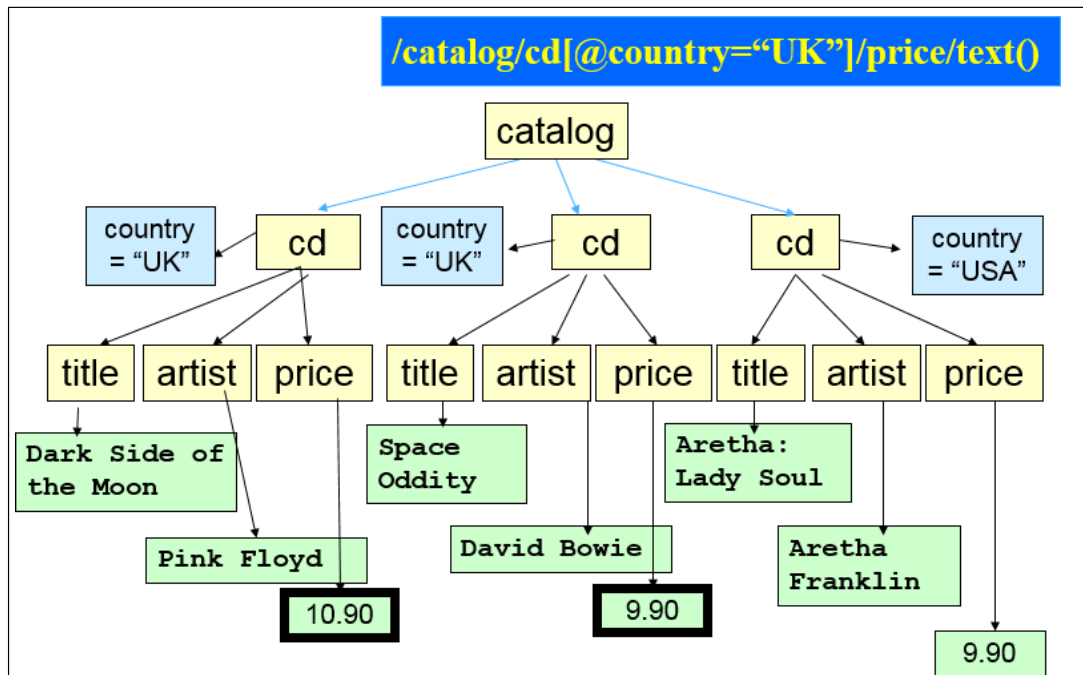


Figure 37 – How to use attributes within conditions to extract price using the text function

Source: King (2014)

## Concluding remarks

This appendix discussed XML syntax and structure. It also examined different schemas as well as how to use XPath to extract and display data.



# Appendix D

This appendix should be referenced when studying the content and the assessment criteria for the week(s) listed below.

<b>Weeks</b>	11-13
<b>Learning Outcome</b>	LO1 & LO2
<b>Assessment Criteria</b>	1.1, 1.2, 1.3, 2.2, 2.3

## Android Development

### Supplementary reading

DiMarzio, J.F. (2017) *Android Programming with Android Studio*, Fourth Edition. John Wiley and Sons, Inc. ISBN:978-1-118-70559-9.

Morrison, M. 2006. *Sams teach yourself XML in 24 hours*. Indianapolis: Sams.

Steele, J. & To, N. 2011. *The Android developer's cookbook: building applications with the Android SDK*. New Jersey: Addison-Wesley.

### Introduction

The Android operating system is an open-source operating system primarily used in mobile devices. Written in Java and based on the Linux operating system, Android was developed by Android Inc. and eventually purchased by Google in 2005. Operating systems for mobile devices are based on a modified version of Linux kernel 2.6. Android code was released by Google under the Apache Licence, which is also a free open-source software.

The Android operating system consists of numerous Java applications and core libraries running under the Java-based Object-oriented Application Framework and Dalvik Virtual Machine (VM). Dalvik is integral for Android to run on mobile devices as these systems are constrained in terms of processor speed and memory. Android has an integrated web browser, SQLite database, Global System for Mobile Communication (GSM) telephony, built-in support for audio, video and still media formats, and support for devices, such as cameras, accelerometers, etc.

## 1.1 Android architecture

Figure 38 illustrates the Android architecture:

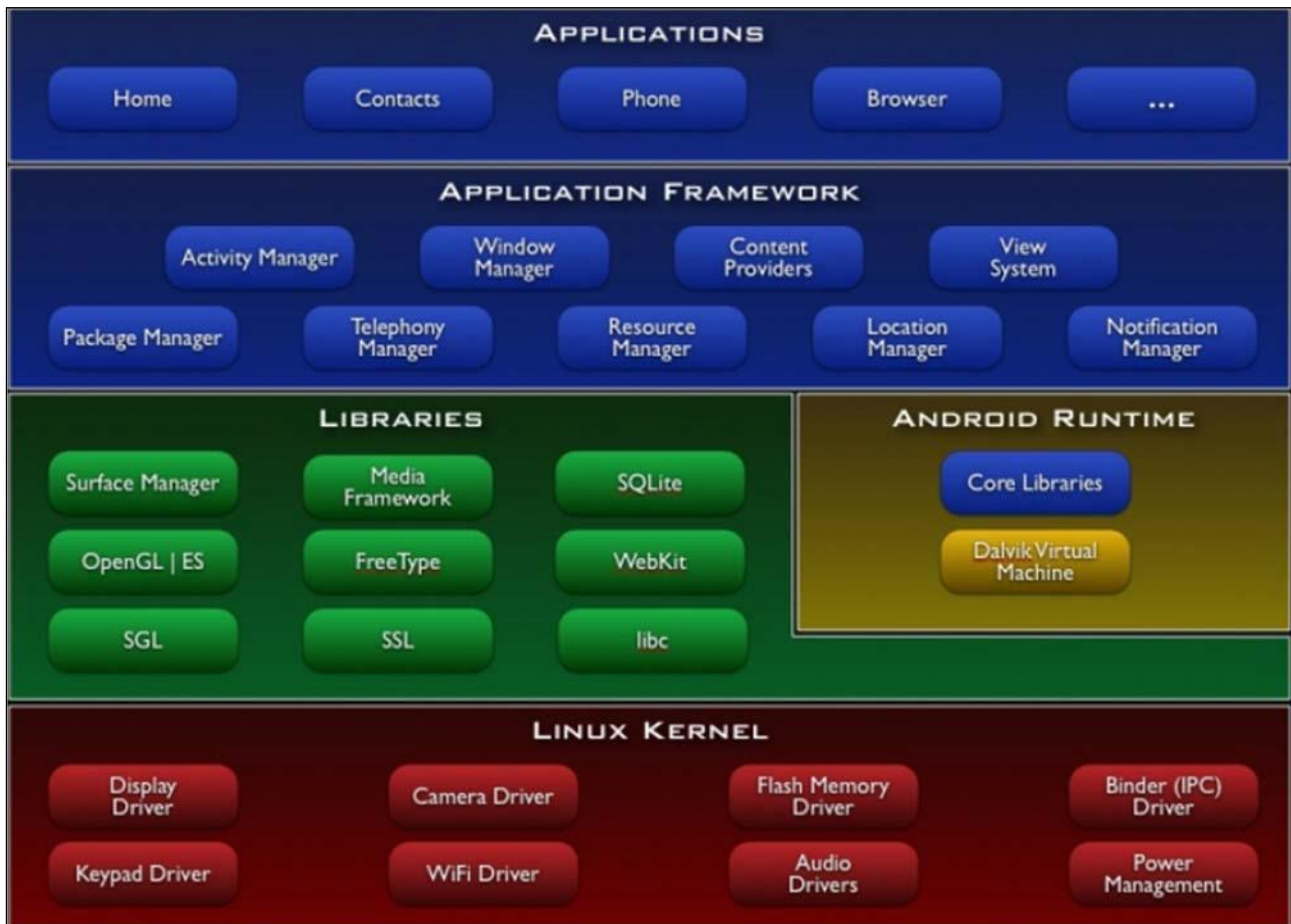


Figure 38 – Android architecture

Source: King (2014)

## 1.2 Linux kernel

At the bottom of the Android architecture structure is Linux 2.6 with approximately **115** patches. This provides basic system functionality, such as process management, memory management and device management, such as cameras, keypads, displays, etc. The kernel also handles all items that Linux is good at, such as networking and a vast array of device drivers, which takes the pain out of interfacing to peripheral hardware.

## 1.3 Libraries

On top of the Linux kernel is a set of libraries: the open-source web browser, WebKit; the well-known library, libc; the SQLite database (a useful repository for storing and sharing application data); libraries to play and record audio and video; secure socket layer (SSL) libraries responsible for Internet security, etc.

Table 1 summarises some of the Android libraries that can be used for Android development:

Library name	Function
<code>android.app</code>	This library provides access to the Application Model; such is the cornerstone of all Android applications
<code>android.content</code>	This library facilitates content access; such publishes and messages between applications and application components
<code>android.database</code>	This library accesses data published by content providers; such includes the SQLite database management classes
<code>android.graphics</code>	A low-level 2D graphics-drawing API that includes colours, points, filters, rectangles and canvases
<code>android.hardware</code>	This library presents an API that provides access to hardware, such as accelerometers and light sensors
<code>android.os</code>	This library provides applications with access to standard operating system services, including message system services and interprocess communication
<code>android.provider</code>	A set of convenience classes that provide access to standard Android content provider databases, such as those maintained by the Calendar and Contact applications
<code>android.text</code>	This library renders and manipulates text on device displays
<code>android.util</code>	A set of utility classes used to perform tasks, such as string and number conversion, XML handling and date and time manipulation
<code>android.view</code>	The fundamental building blocks of application user interfaces

Table 1– Android libraries

Source: [http://www.tutorialspoint.com/android/android\\_tutorial.pdf](http://www.tutorialspoint.com/android/android_tutorial.pdf)

## 1.4 Runtime

This is the third section of the architecture, available on the second tier from the bottom. Such provides a key component, called the '**Dalvik VM**', which is a kind of JVM specially designed and optimised for Android. '**Dalvik**' is the name of the **Android JVM**. It is non-standard, register-based, rather than stack-based; such uses non-standard byte code and is a tool used to convert Java class files to Dalvik dex files. It does not use a just-in-time (JIT) compiler.

The **Dalvik VM** makes use of core Linux features, such as memory management and multithreading, which is intrinsic to the Java programming language. The Dalvik VM enables

every Android application to run in its own process with its own instance of the Dalvik VM. Runtime also provides a set of core libraries, which enable Android application developers to write Android applications using standard Java programming language.

## 1.5 Application Framework

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

## 1.6 Applications

Application services encompass the following:

- **Views:**

These are standard sets of component for graphics user interfaces (GUIs).

- **Content providers:**

These are standard interfaces for sharing data between different applications.

- **Resource Manager:**

This accesses localised strings, graphics, etc.

- **Notification Manager:**

This provides access to custom alerts.

You will find all Android applications at the top layer of the Android architecture. You will write your application that is to be installed on this layer **only**. Examples of applications are Contacts, Books, Browser, etc. Application components are essential building blocks of Android applications. These components are loosely coupled by the application manifest file **AndroidManifest.xml** that describes each component of an application and how such interacts.

There are four main components that can be used within Android applications (Steele & To, 2011:23):

1. **Activities:**

Such dictate the user interface and handle user interaction with a smartphone screen, e.g. playing a game.

2. **Services:**

Such handle background processing associated with an application, e.g. playing music.

3. **Broadcast receivers:**

Such handle communication between the Android operating system and an application, e.g. triggering an alarm.

#### 4. Content providers:

Such handle data and database management issues, e.g. opening a phone contact.

### 1.7 Activities

An **activity** represents a single screen with a user interface. For example, an e-mail application might have one activity to show a list of new e-mails, another **activity** to compose an e-mail and yet another activity to read an e-mail. If an application has more than one **activity**, one of them should be marked as the activity that is presented when said application is launched. An **activity** is implemented as a subclass of the **Activity** class (Figure 39):

```
public class MainActivity extends Activity {  
  
    //code goes here  
  
}
```

Figure 39 – Declaring activities

Figure 40 illustrates a diagrammatical representation of the **Activity** life cycle:

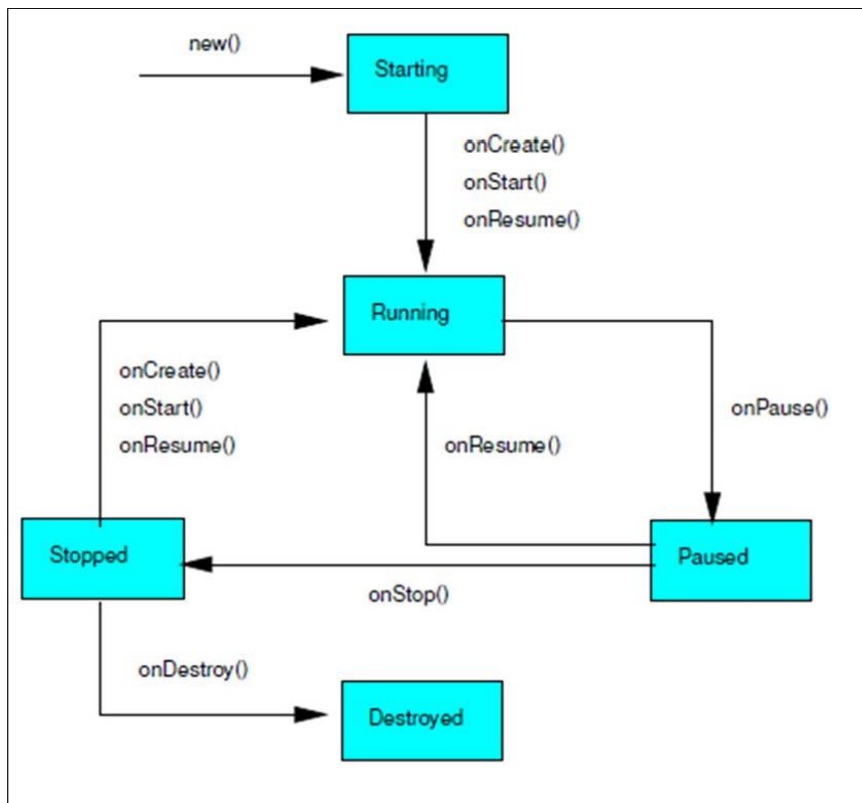


Figure 40– Diagrammatical representation of the Activity life cycle

Source: King (2014)

### 1.7.1 Activity callbacks

Callbacks occur at each change of state:

#### Starting state:

- This encompasses the initialisation and creation of an application.

#### Running state:

- This is what is currently on screen and interacting with the user (only one application at a time can be in this state).

#### Paused state:

- This is what may still be on the screen but such is not interacting with the user.

#### Stopped state:

- This is no longer visible but still in memory.

#### Destroyed state:

- This is no longer in memory.

Activity callbacks are as follows:

#### **onCreate():**

- This is as an application is created with saved state passed as the parameter.

#### **onStart():**

- This is as an application starts activity.

#### **onPause():**

- This is when an application loses focus (you may wish to reduce activity and store state).

#### **onResume():**

- This is the restoration of state.

#### **onStop():**

- This activity is no longer visible.

#### **onDestroy():**

- This activity is about to be destroyed.

#### Note

All callbacks should start with **super.callback()** to ensure that activities perform the correct housekeeping.

- **StartActivity** calls specifying an intent. An **intent** is the message that is passed between components such as activities, content providers, broadcast receivers, services etc
- Intent can indicate a type of activity, e.g. send e-mail, or an explicit activity (for when you are working within your own application).
- Active activities form a stack: the **back stack**.
- Only the top of a stack is running.
- **Back key** pops a stack; it deletes the top activity and resumes the new top of stack.
- It is possible to have multiple instances of an activity on stack at any one time.

## 1.8 Services

A **service** is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application or it might fetch data over the network without blocking the user's interaction with an activity. A service is implemented as a subclass of the Service class (Figure 41):

```
public class MyService extends Service {  
  
}
```

Figure 41 – Declaring services

Figure 42 illustrates a diagrammatical representation of the Service life cycle:

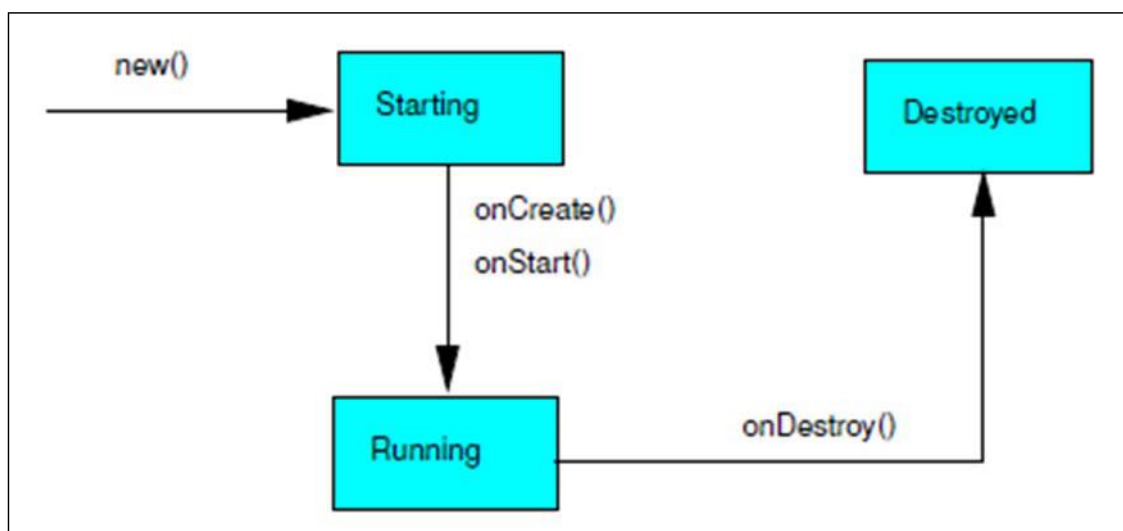


Figure 42 – Diagrammatical representation of the Service life cycle

Source: King (2014)

## 1.9 Broadcast receivers

A **broadcastReceiver** simply responds to broadcast messages from other applications or from the system. For example, applications can initiate broadcasts to let other applications know that some data has been downloaded to a device and is thus available for them to use; a broadcast receiver will intercept this communication and initiate the appropriate action. A

**broadcastReceiver** is implemented as a subclass of the **BroadcastReceiver** class; each message is broadcast as an intent object.

### 1.10 Content providers

A **contentProvider** supplies data from one application to another upon request. These requests are handled by the methods of the **ContentResolver** class. Data may be stored in a file system, database or somewhere else entirely. A **contentProvider** is implemented as a subclass of the **ContentProvider** class; such must implement a standard set of APIs that will enable other applications to perform transactions (Figure 43):

```
public class MyContentProvider extends ContentProvider {  
  
}
```

Figure 43 – Declaring contentProviders

Figure 44 illustrates a diagrammatical representation of ContentProvider:

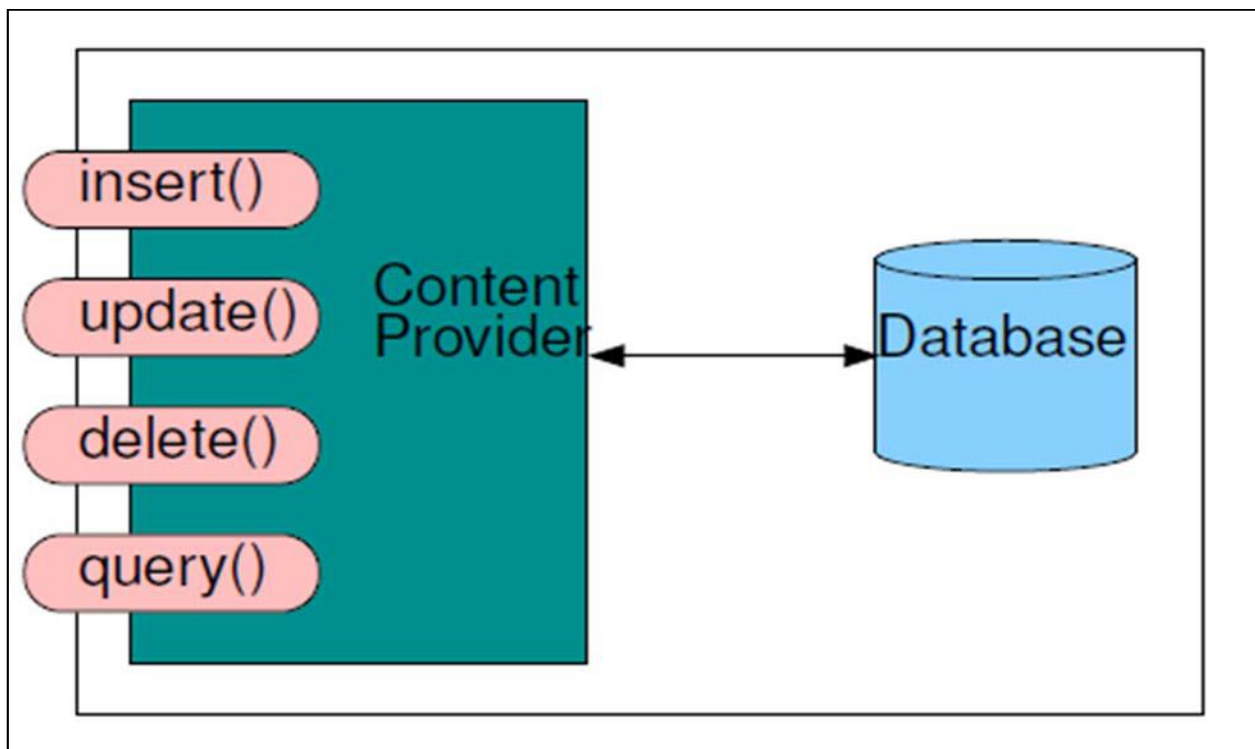


Figure 44 – Diagrammatical representation of ContentProvider

Source: King (2014)



Content providers allow you to share data between applications. They also avail the CRUD (create, read, update and delete) Interface.

Here is a list of methods which you will need to override in the **ContentProvider** class to have such working:

**onCreate():**

- This method is called when a provider is started.

**query():**

- This method receives a request from a client; the result is returned as a cursor object.

**insert():**

- This method inserts a new record into a provider.

**delete():**

- This method deletes an existing record from a provider.

**update():**

- This method updates an existing record from a provider.

**getType():**

- This method returns a multi-purpose Internet mail extension (MIME) type of data at the given URI.

## 1.11 Directory structure

**Table 2** summarises the directory structure of Android:

User-generated file	Description
src/	This contains the Java packages that developers write or import for an application; each package can have multiple <b>.java</b> files
res/layout/	This contains the XML files that specify the layout of each screen
res/values/	This contains the XML files used as reference by other files
res/drawable-hdpi/mdpi/ldpi	Directories that contain picture resources that an application can use; its high, medium and low dots-per-inch resolutions respectively
assets/	This contains additional non-media files that an application can use
AndroidManifest.xml	This specifies a project to the Android operating system

Auto-generated file	Description
gen/	This contains auto-generated code, such as <b>R.java</b>
default.properties	This contains project settings

**Table 2 – Android directory structure**

Source: Steele and To (2011:27)

## 1.12 Android Development

Android application development can be executed on either of the following operating systems:

- Microsoft Windows XP or later version
- Mac OS X 10.5.8 or later version with an Intel chip
- Linux (including GNU C Library 2.7) or later version

All the required tools to develop Android applications are freely available and can be downloaded from the web. The following is a list of software that you will need before starting your Android application programming:

- Windows 10 Computer
- Java JDK 8
- Android Studio 3.6 or Higher

### 1.12.1 Android Studio Introduction and Installation

Android developers write and test their applications on computers and then deploy these onto a mobile device for further testing. In this section, we look at how the setup is done and explore Android Studio.

Android Studio is an integrated development environment (IDE) for Android. It is the official Google IDE used in Android development. Android Studio is built on IntelliJ IDEA software which is a Java IDE used in software development. Android Studio replaced the Eclipse Android Development Tools (ADT) to become the primary IDE in the development of Android apps.

Although NetBeans supports Android development, many developers use Android Studio for writing mobile apps owing to its ease of use and the many features it has to offer. Android Studio provides a graphical user interface (GUI) like any other IDE like NetBeans and it supports other programming languages apart from Java. To enable this support add plug-ins that provide the specific functionalities you need.

As you are using Android Studio to write Android apps, Android Studio will be using the Android Software Development Kit (SDK) to create, debug and run Android applications. The SDK will be used to keep the Android Studio updated with every new release of Android. In developing Android apps, the Minimum Required SDK is used to designate to the Google Play Store the oldest version of Android that the app can run on. The Target SDK is the version of Android the app is designed to run on and will compile with. You will know more about this in the sections to follow in this unit.

You will need about 10 GB of free disk space and 4 GB RAM to be able to install Android Studio. If you were using NetBeans and had already installed the Java Development Kit, you will not have to re-install the JDK.

Android Studio uses the Java programming language for developing Android applications so configure your programming environment for Java development. The following software needs to be installed on your computer to develop Android applications:

The JAVA Development Kit (JDK) version 8 is available for download at:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

Android Studio v3.6, which is available for download at:

<https://developer.android.com/studio/index.html>

### 1.12.3 Installing Android Studio on Windows 10

The following steps should be taken on the host computer which will be used for development.

1. Install the Java Development Kit (JDK) 8.

After installing Java, set the **PATH** and **JAVA\_HOME** environment variables to refer to the directory that entails java and **javac**.

Alternatively, you could also right-click on **This PC** and then select Properties >

**Advanced > Environment Variables**. You would update the **PATH** value and press **OK**.

2. After downloading Android Studio, double click on the executable file. Click yes to authorise the Android software to make changes to your computer. You will be presented with the following screen:

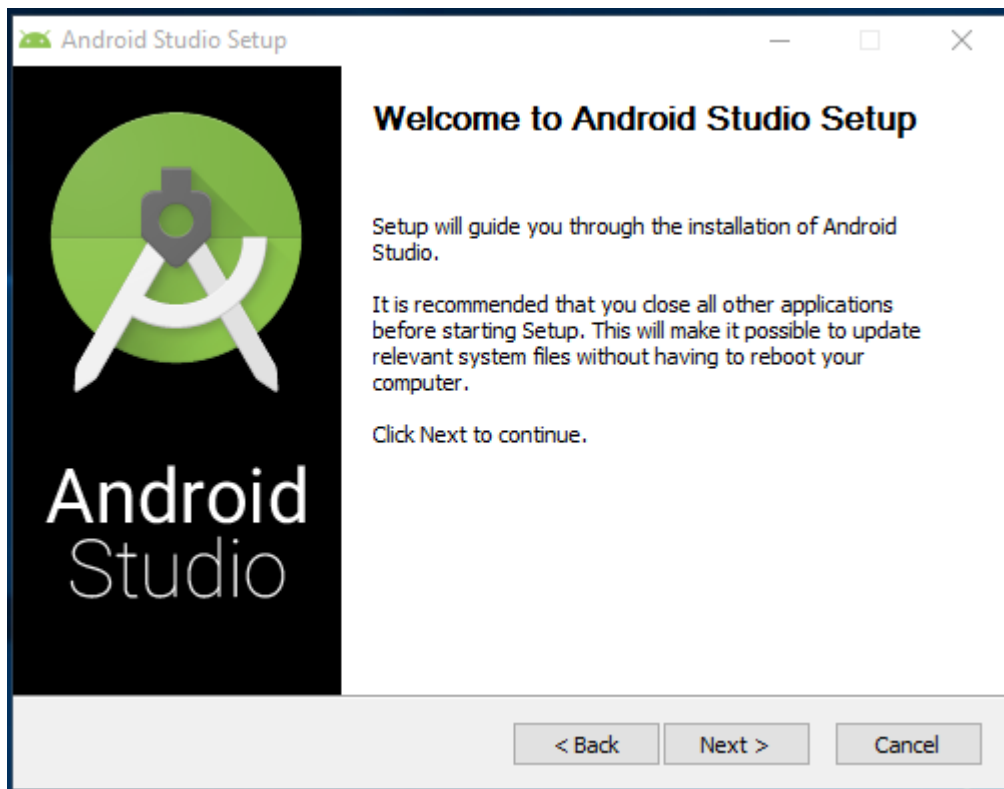
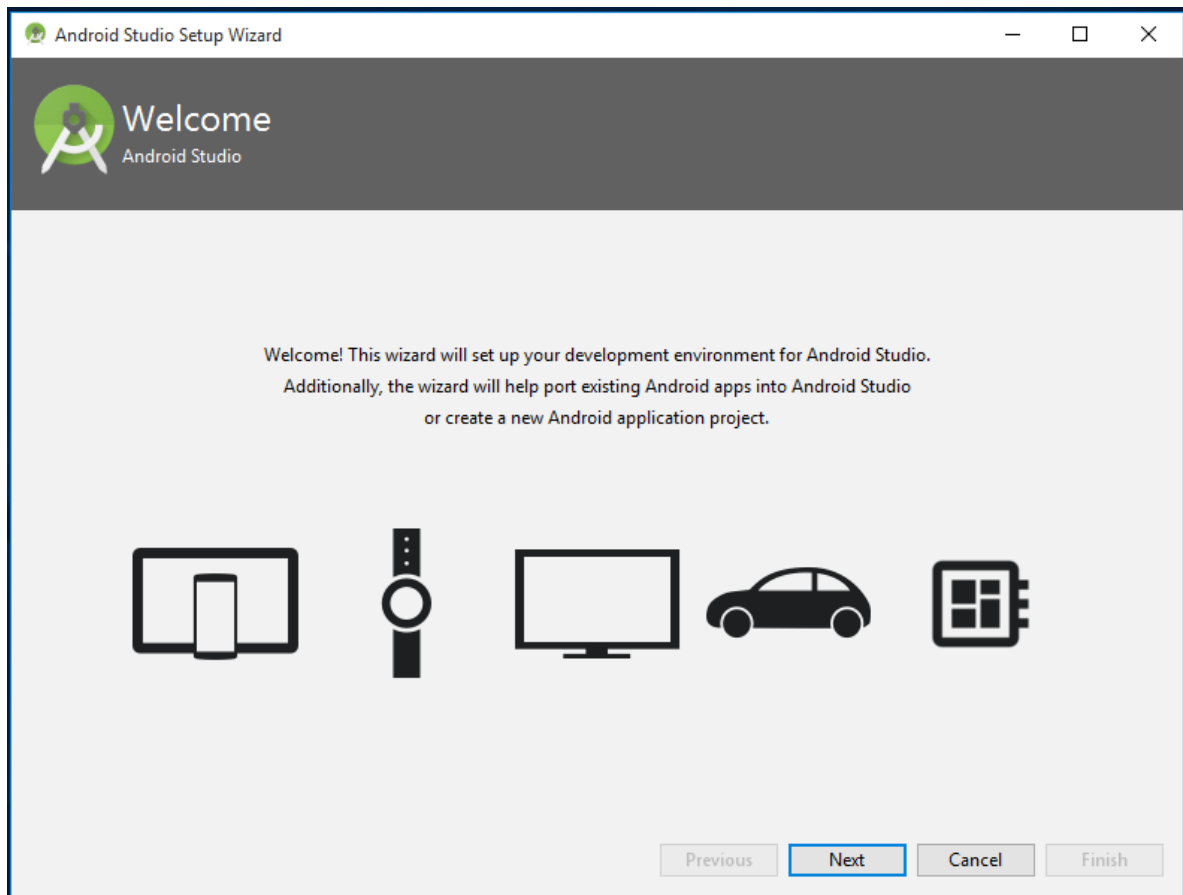


Figure 45 Android Studio Setup

3. Click **Next**.
4. On the following set-up screen, make sure “**Android Studio**” and “**Android Virtual Device**” are ticked in the components to install box. Click **Next** once you are done.
5. Click **Next** to install Android Studio in the default location
6. Click “**Install**” to install Android Studio on your computer.
7. When a Complete Installation dialog box pops out after starting Android Studio, select “**Do not import settings**”.
8. Click **Finish** once it has completed the installation process.  
The Android Studio Setup Wizard will launch. This will help you to set up the development environment for Android Studio.
9. Click **Next**.
10. Choose the **Standard** Install Type, then click **Next**.
11. Select the Theme of your choice. In this unit, we will use the IntelliJ Theme.
12. Click **Finish**.

Android Studio will start to download essential components which include Android SDK Platform, Android SDK Tools and Google repository. The next screen will show you Android Studio downloading the components. This process might take a while depending on the speed of your internet connection.



**Figure 46 Android Studio Setup Wizard**

13. Click **Finish**

14. You will be presented with the Welcome Page of the setup wizard:

Now your environment is now configured and ready for Android application development.

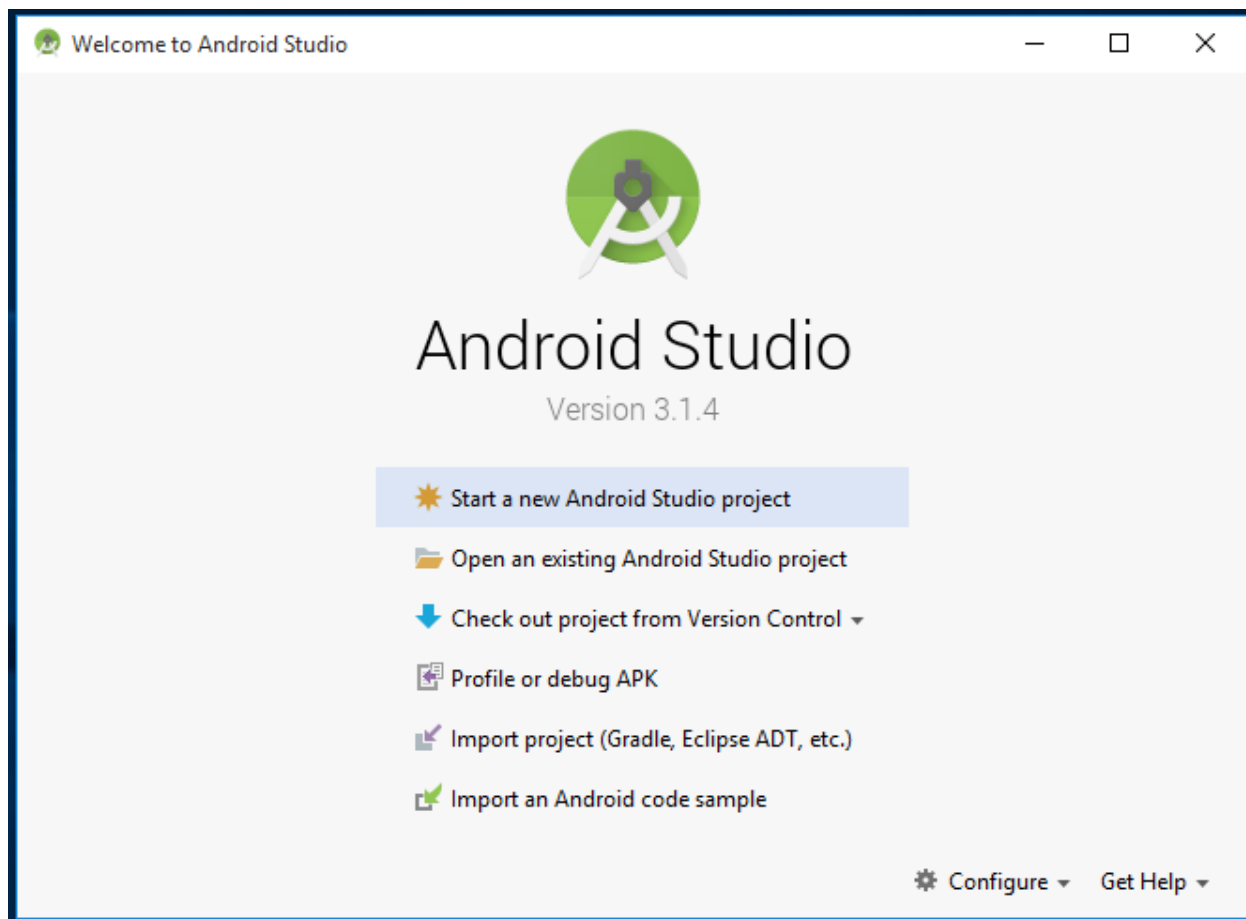


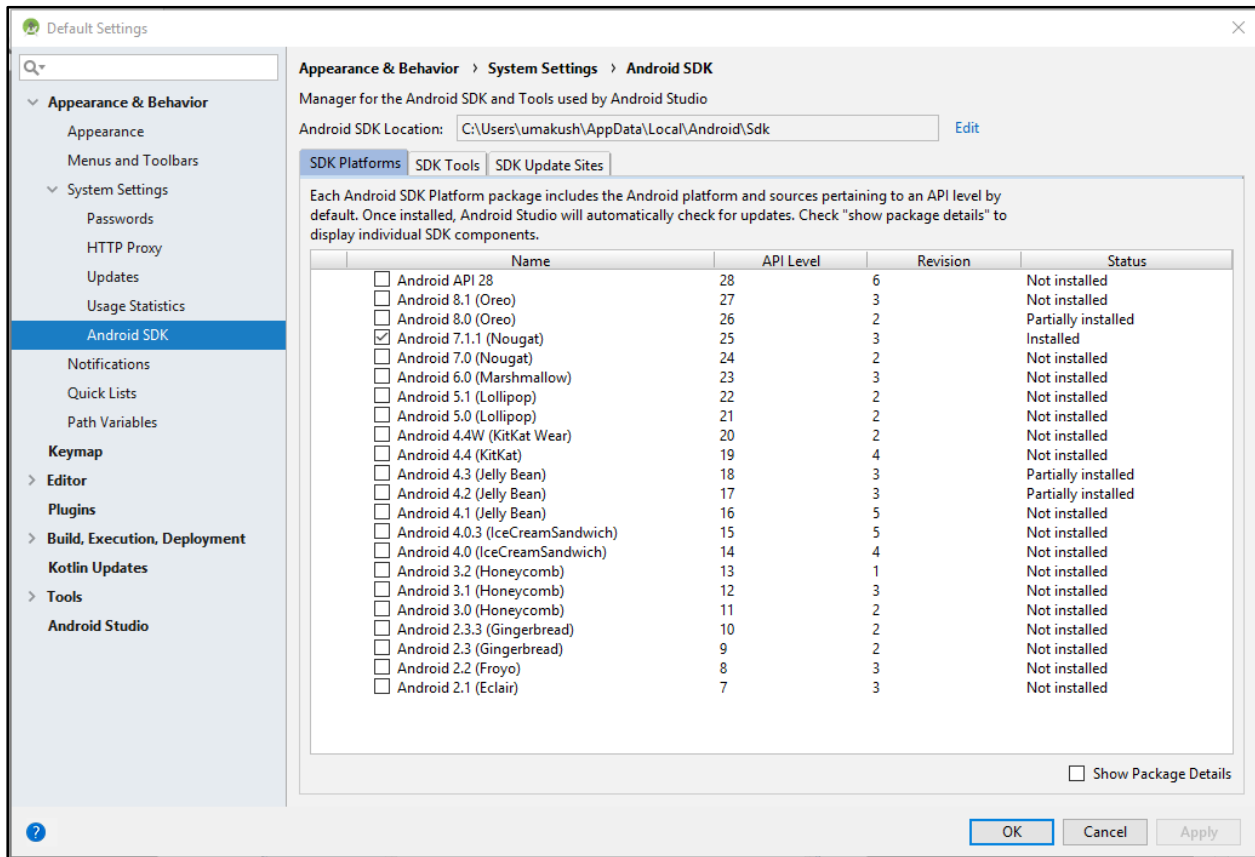
Figure 47 Android Studio Welcome Window

#### 1.12.4 SDK Manager

The Android SDK (Software Development Kit) Manager is used to install plugins, tools and other essential packages necessary for Android Studio to work properly. The SDK Manager gives you access to functions such as compiling, packaging, developing and also deploying Android applications. Install Android SDK via the SDK manager. Android SDK contains all the packages and tools used in developing an Android application.

Open SDK Manager from the Welcome Page by clicking **Configure** on the bottom right of the screen (see Figure 48 below). Alternatively, SDK Manager can be launched within Android Studio by going to **Tools -> Android -> SDK Manager**.

After launching SDK Manager, you will be presented with the following window:



**Figure 48 - SDK Manager**

On the left are the various settings you can configure for Android Studio. From Figure 48 above, it can be seen the Android SDK installed is Android 7.1.1 Nougat. If you would like to install a different SDK, tick the box next to the SDK platform package you would like to install and then click **Apply**.

#### 1.12.4.1 Emulator and virtual devices

In an earlier section you installed Android Studio. Android Studio contains an Android device emulator which is a software application that imitates another device, such as a computer or mobile device. The Android device emulator is then used to run the Android Virtual Device (AVD) to emulate a real Android device.

The main purpose of AVDs is to simulate Android applications without using a real Android device. With an AVD you can test your application on different device settings and versions without necessarily having the device in hand. The advantage of this is when defining the AVD you can give it your own specifications, such as the version of your Android API or the resolution.

AVDs are very flexible and powerful in the sense that you can start multiple devices at once. These can have different configuration settings, which makes it easier and quicker for you to test your application on different devices all at once.

#### 1.12.4.2 Emulator speed

There are two types of AVDs that you can create, namely an Android device or a Google device. If you choose to create an AVD for Android then it will have to contain the programs from the Android Open Source Project. However, if you decide to choose to create an AVD for Google it will contain additional Google-specific code that will allow you to test applications which use Google Play.

#### 1.12.4.3 Creating a simple hello world app and AVD

To create an AVD, first create an Android project. Follow these steps:

1. Open Android Studio from the **Start menu** -> **All Apps** -> **Android Studio**. You will be presented with the Welcome Screen as shown below.

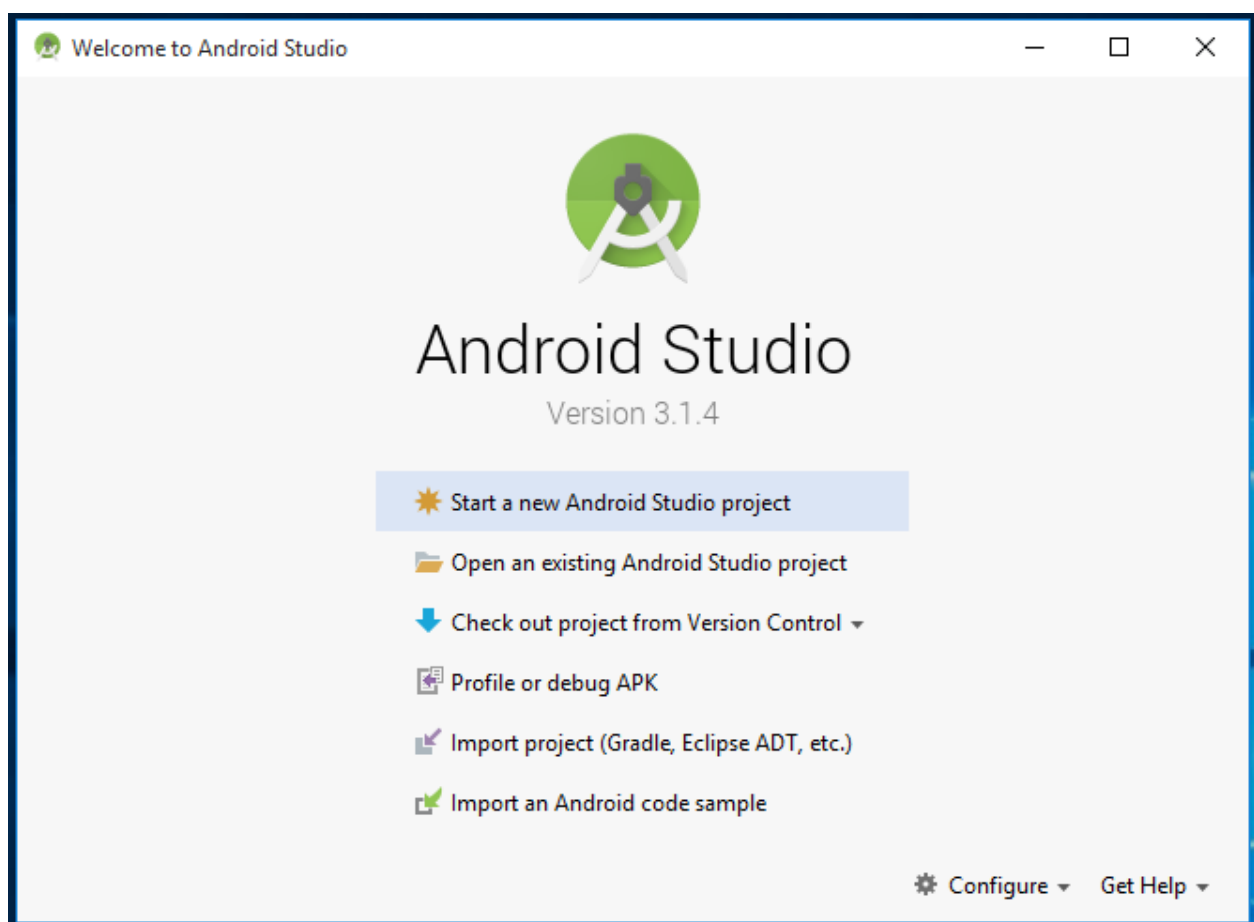


Figure 49- Android Studio Welcome Page

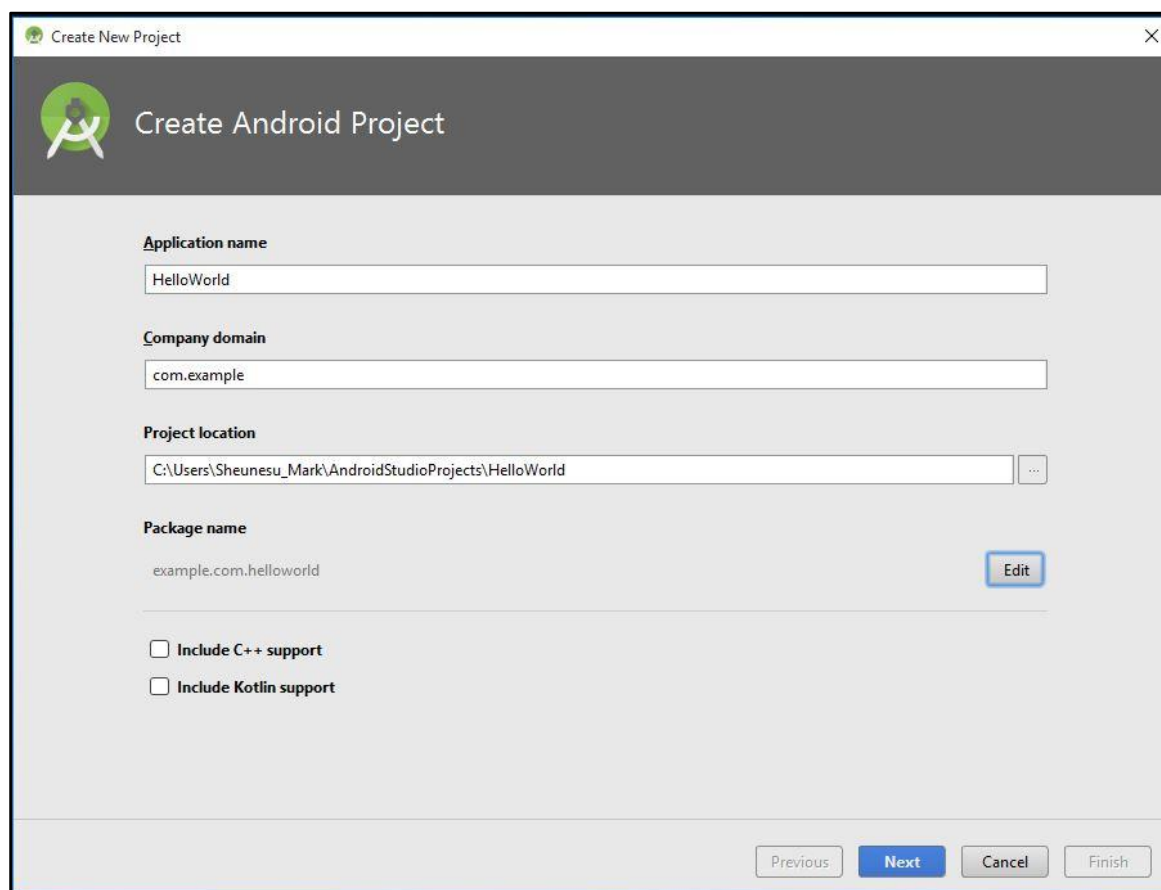


Before creating a new project, you have to know the following about your application:

- The features of your application
- The activities required

Let us create a simple “Hello World” application. It displays **Hello World** to the user so you can get the feel of programming in Android Studio.

2. To create a new project, Click “**Start a new Android Studio project**”. This displays a new Android Project-creation screen. Type HelloWorld as the application name and com.example as the Company Domain. You will notice the PackageName is generated automatically but it can be edited.
3. Click **Next**.



**Figure 50- Creating an Android Project**

On the Target Window, if you want to select *another target* under *Phone and Tablet*, choose from the drop-down menu but preferably leave it as the default **API 15: Android 4.0.3 (IceCreamSandwich)**.

4. Click **Next**.

5. Select “**Empty Activity**”, then click **Next**.

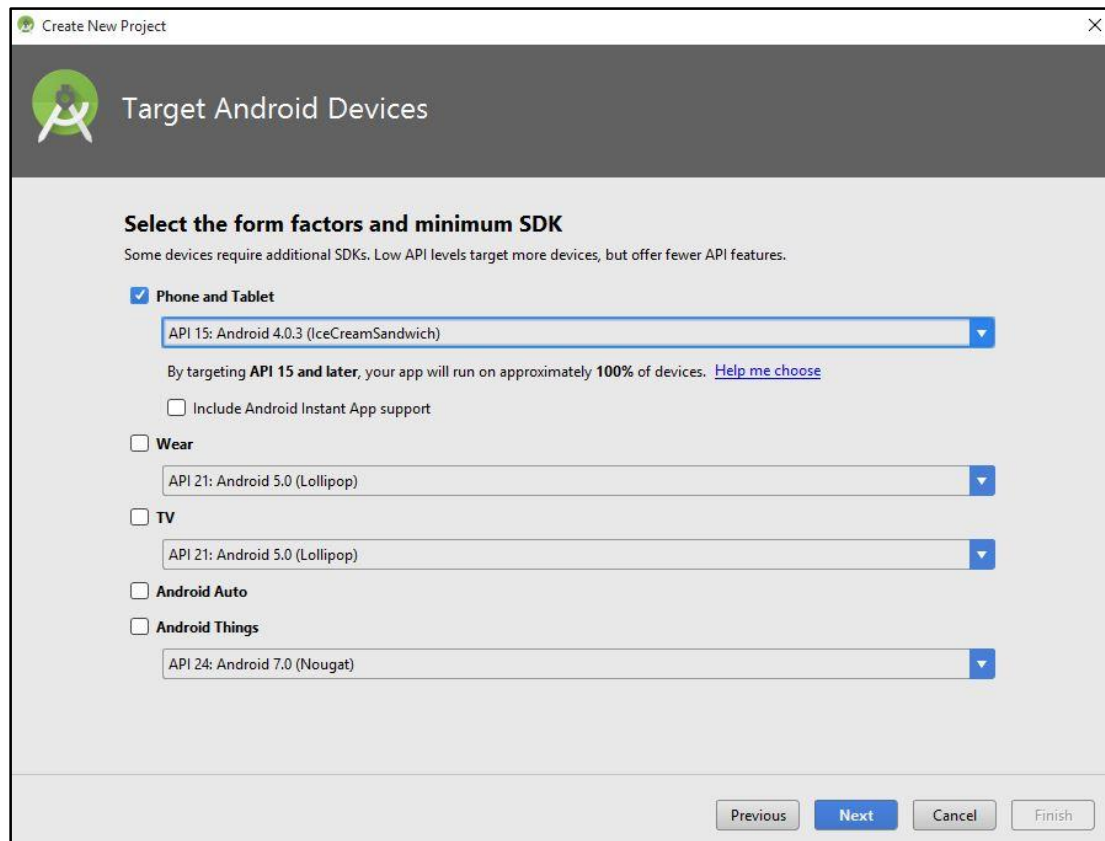


Figure 51- Target System

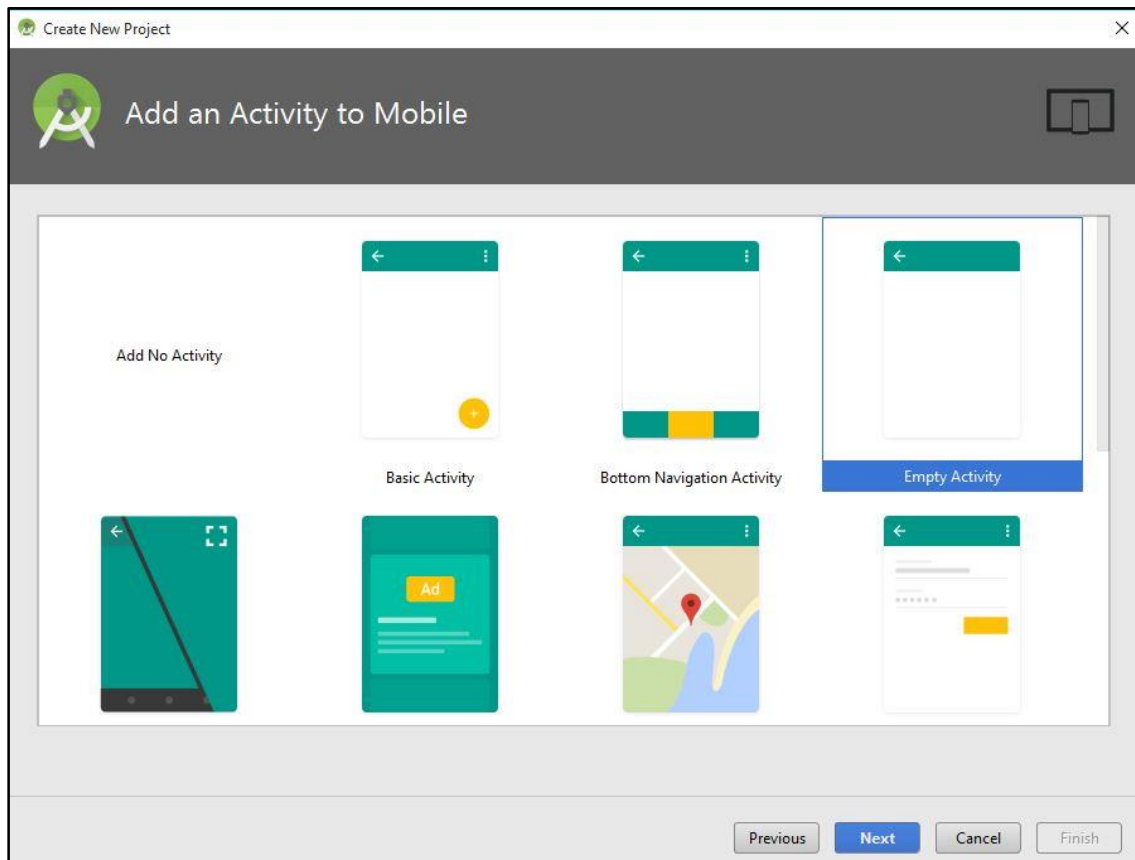
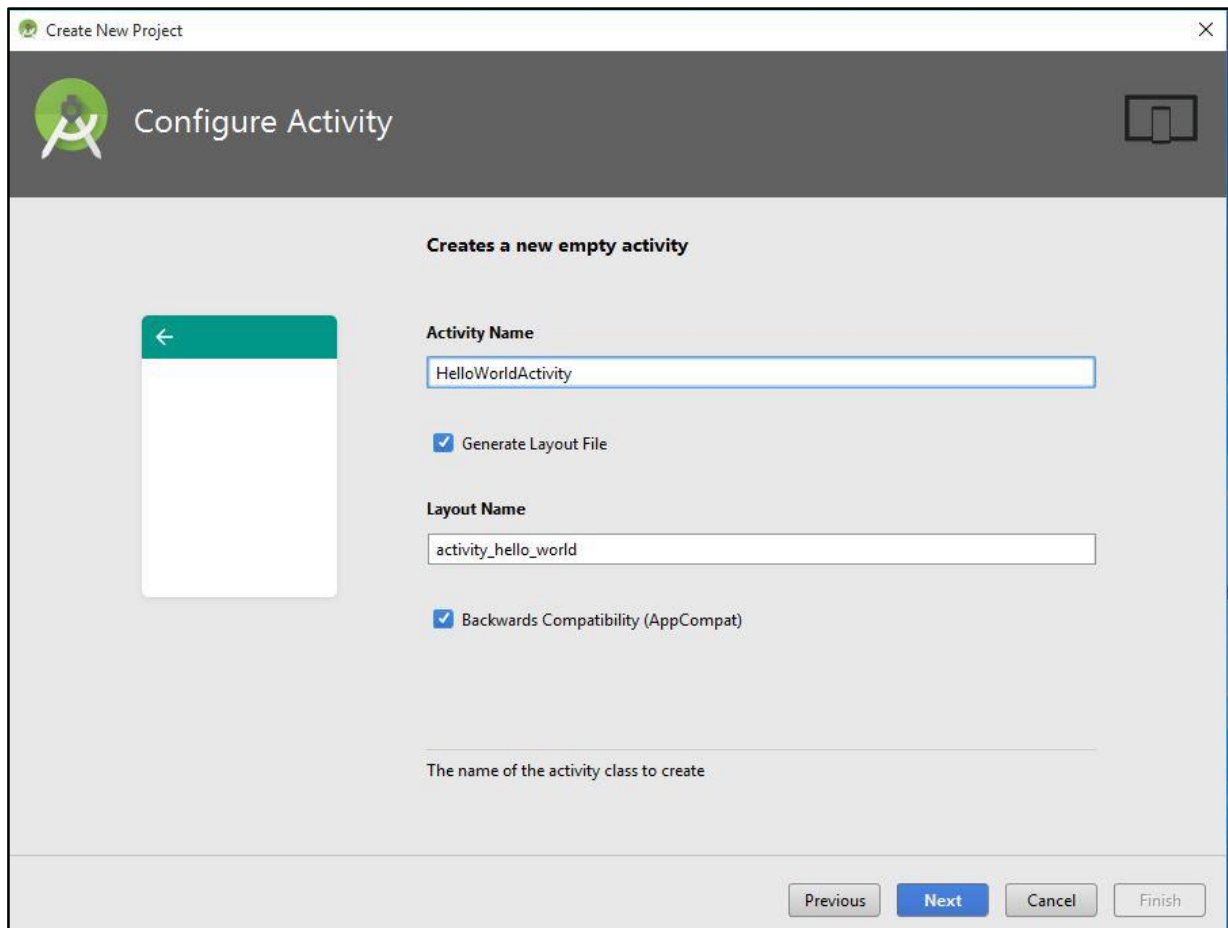


Figure 52- Activity selection

6. Type **HelloWorldActivity**, as the **ActivityName**, then leave the rest and click **Next**.



The screenshot shows the 'Configure Activity' dialog box in Android Studio. The title bar says 'Create New Project'. The dialog has a dark header with the Android Studio logo and the text 'Configure Activity'. Below the header, there's a section titled 'Creates a new empty activity'. On the left, there's a preview of a mobile app screen with a green header bar and a white body. On the right, there are two text input fields: 'Activity Name' with the value 'HelloWorldActivity' and 'Layout Name' with the value 'activity\_hello\_world'. There are two checkboxes: 'Generate Layout File' (checked) and 'Backwards Compatibility (AppCompat)' (checked). At the bottom, there's a label 'The name of the activity class to create' and four buttons: 'Previous', 'Next' (highlighted in blue), 'Cancel', and 'Finish'.

Figure 53- Writing Activity Name

7. Click **Finish**.

You will be presented with the Android Studio IDE.

#### 1.12.4.4 Android Studio Folder Structure

Before you run your application, you should be aware of a few directories and files in the Android project (Figure 54):

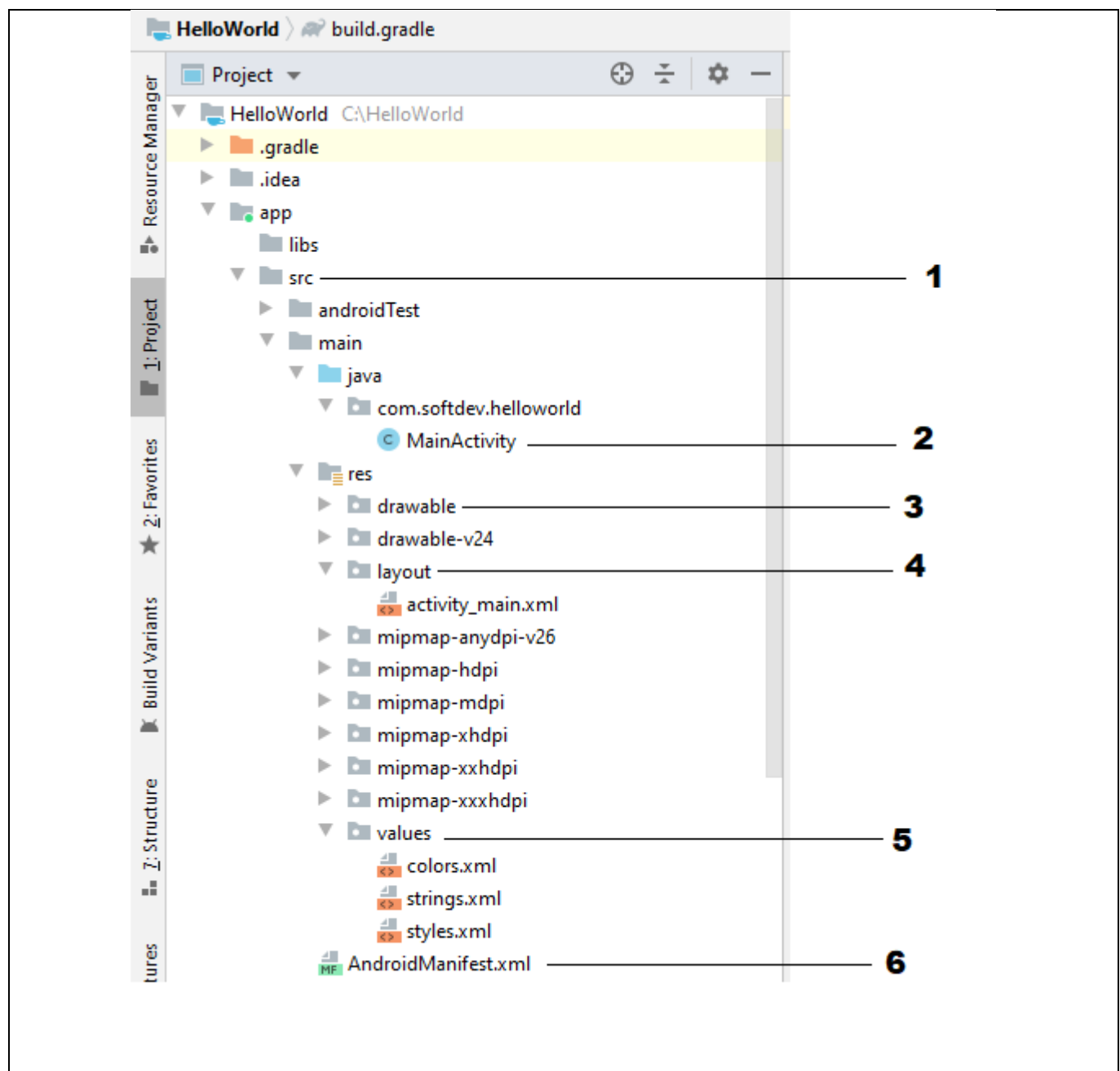


Figure 54 - Android Studio Project Structure

Number	Folder	Description
1	src	This contains the <b>.java source files</b> for your project. By default, it includes the <b>MainActivity.java source</b> file that has an <b>Activity</b> class that runs when your application is launched using the application icon

2	MainActivity	This is the Activity class of your project. It gets automatically generated the time a new project is created. Each activity file, has an associate XML file located in the layout folder
3	res/drawable	This is a directory for drawable objects that are designed for high-density screens
4	res/layout	This is a directory for files that define your application's user interface
5	res/values	This is a directory for other various XML files that contain a collection of resources, such as strings and colour definitions
6	AndroidManifest.xml	This is the <b>manifest file</b> , which describes the fundamental characteristics of the application and defines each of its components

A folder named HelloWorld should be automatically added to the Project Explorer. Click the little arrow on the left of the HelloWorld Project Folder, helloWorld/app/src/main/java/example/com/helloworld/, to expand it and then double-click the **MainActivity.java** file.

A code similar to the following will be displayed:

```
package example.com.helloworld;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class HelloWorldActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_hello_world);
    }
}
```

**Figure 55 - MainActivity.java**

This is the code generated when you open the **MainActivity.java** file. This class is responsible for the interface of the application, i.e. this is the activity that is going to display the “Hello World” Text. This is what Java code looks like. The `onCreate()` method is one of many that are fired when an activity is loaded. Make no changes to the code.

**Note**

To expand the import section from the code above, click the little **+** circle on the left close to the import statement.

Now, alter the **strings.xml** file and expand the res folder from the main folder, `res/values/strings.xml`:

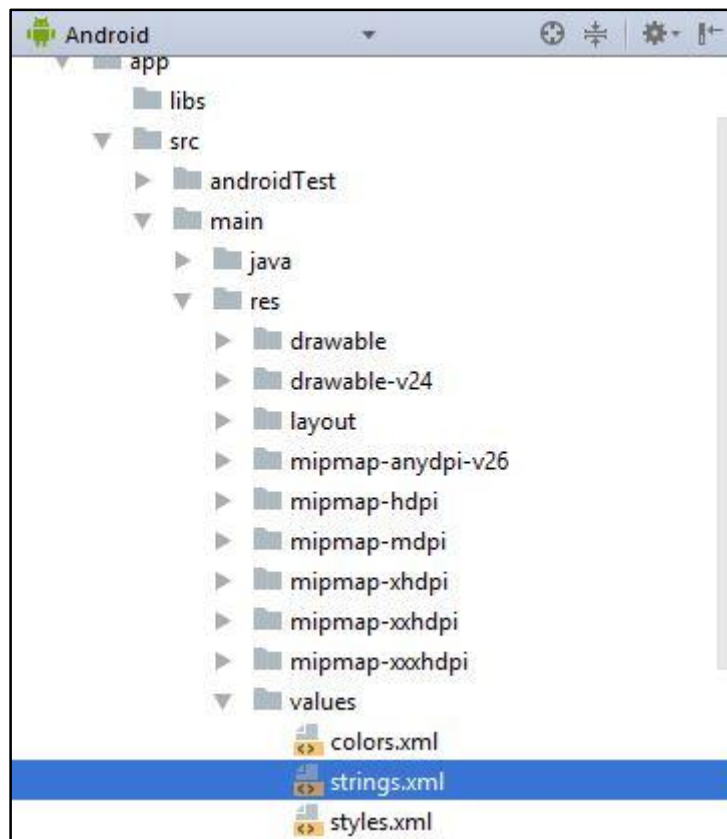


Figure 56 - strings.xml

#### 1.12.4.4 Working with strings

Whenever you want to display text in your application, use string resources. ` string resources with the `<string>` tag and save the file, for example, if you want to display your name, you tag it like this:

```
<string name="name">Sheunesu Makura</string>
```

First, give the string a name and write the value of the string within the `<string>` tags.

Let us look at the following strings.xml code for the HelloWorld project:

```
<resources>
    <string name="app_name">HelloWorld</string>
</resources>
```

Figure 57 - strings.xml

The string from the code is the name of the application. Add another string “hello”. This string will display the “HelloWorld” message.

Your code should be similar to the following:

```
<string name="hello">Hello World</string>
<string name="app_name">Hello World</string>
```

Save the file after you have made the changes.

#### 1.12.4.5 Android Manifest File

Whatever component you develop as part of your application, you must declare all of its components in a **Manifest** file, called ‘**AndroidManifest.xml**’, which resides at the root of the application project directory. Said file works as an interface between the Android operating system and application; if you do **not** declare your component in this file, it will **not** be considered by the operating system.

For example, a default **Manifest** file will look like Figure 56:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.softdev.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
```

```

        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>

```

Figure 58 – Manifest directory

The `<application>...</application>` tags enclose the components related to the application. Attribute `android:icon` points to the application icon available under `res/drawable`. The application uses the image named `ic_launcher.png` located in the **drawable** folders.

The `<activity>` tag is used to specify an activity and the `android:name` attribute specifies the fully qualified class name of the **activity** subclass. The `android:label` attribute specifies a string to use as the label for said activity. You can specify multiple activities using `<activity>` tags.

The action for the **intent-filter** is named `android.intent.action.MAIN` to indicate that this activity serves as the entry point for the application. The category for the **intent-filter** is named `android.intent.category.LAUNCHER` to indicate that the application can be launched from a device's launcher icon.

`@string` refers to the **strings.xml** file. `@string/app_name` refers to the `app_name` string defined in the **strings.xml** file, which is **"HelloWorld"**. Similarly, other strings are populated in the application. The following is a list of tags, which you will use in your **Manifest** file to specify different Android application components:

- `<activity>`: activity elements
- `<service>`: service elements



- `<receiver>`: broadcast receiver elements
- `<provider>`: content provider elements

#### 1.12.4.5 Working with layouts

Design and preview layouts of your application from the layout resources. From the project folder expand `res/layout/activity_hello_world.xml` or `activity_mail.xml`

Your `activity_hello_world.xml` layout file should be similar to the following:

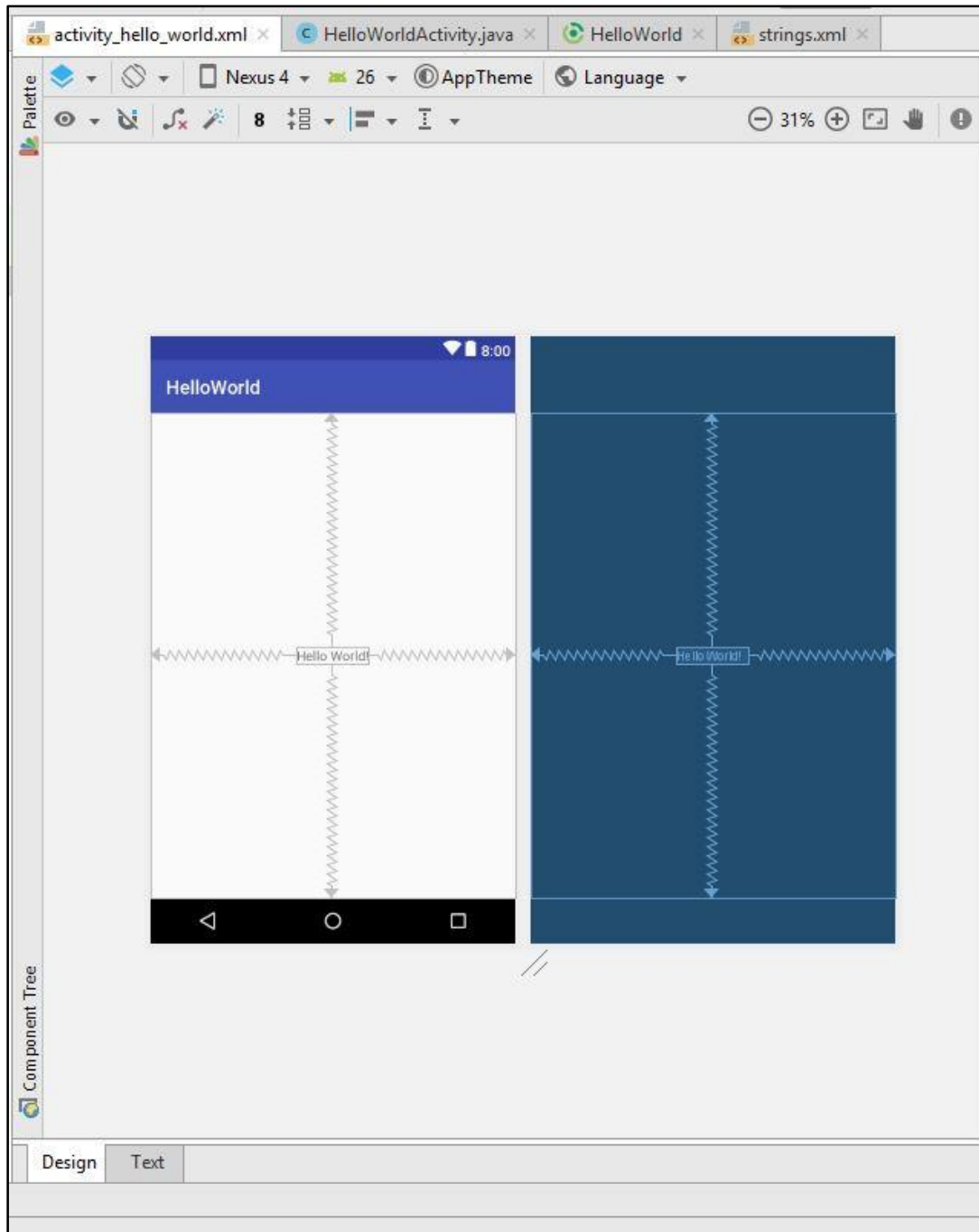


Figure 59 - `activity_hello_world.xml`

If you look at the interface (emulator), your application name and the “Hello World” text is displayed. This is how your application is going to look if you run it. On the left of the emulator on “Palette” are all the controls you will need to create your interface. Simply drag and drop controls to the interface or add these programmatically. In this case make no changes to the file.

The **activity\_main.xml** is a layout file available in the res/layout directory that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application.

Click the stacked icon on the top right corner to view the code for the layout. You should see something similar to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Figure 60 - activity\_main.xml

`TextView` is an Android control used to build GUIs; it has various attributes, such as `android:layout_width`, `android:layout_height`, etc., which are being used to set width, height, etc.

#### 1.12.4.6 Launching the application

The emulator launches a window that looks like an Android phone and runs actual ARM instructions. The initial startup of the emulator is slow even on high-performance computers. The emulator can be configured in many ways to make it emulate many aspects of a real Android device, such as sending SMSes, making phone calls and effecting screen-orientation changes. The target virtual device must be created before the emulator can properly run.

To install and debug Android applications on Android devices, configure your operating system to access the phone via USB cable (for Mac OS no configurations are needed). However, for Windows install the appropriate USB drivers for your device which are available for download at: <http://developer.android.com/sdk/win-usb.html>.

On the main menu, click **Run -> Run App** to deploy the Hello World App which we created.

A window will appear which asks you to select a deployment target. We did not create one before so to create a device where our app will run, click “**Create New Virtual Device**”.

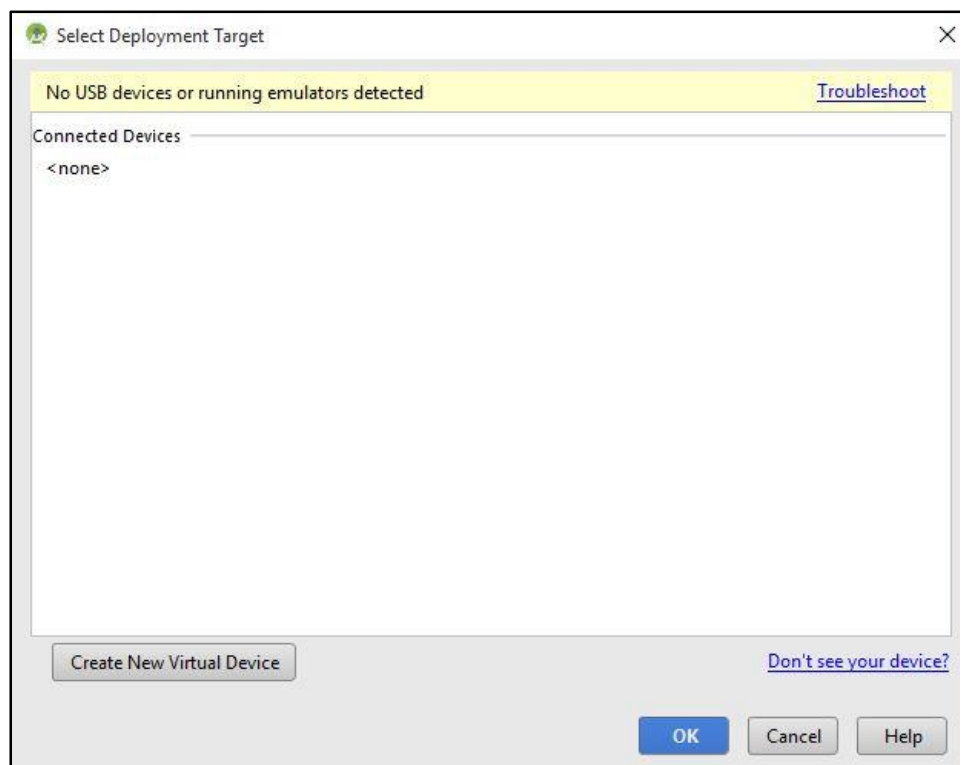
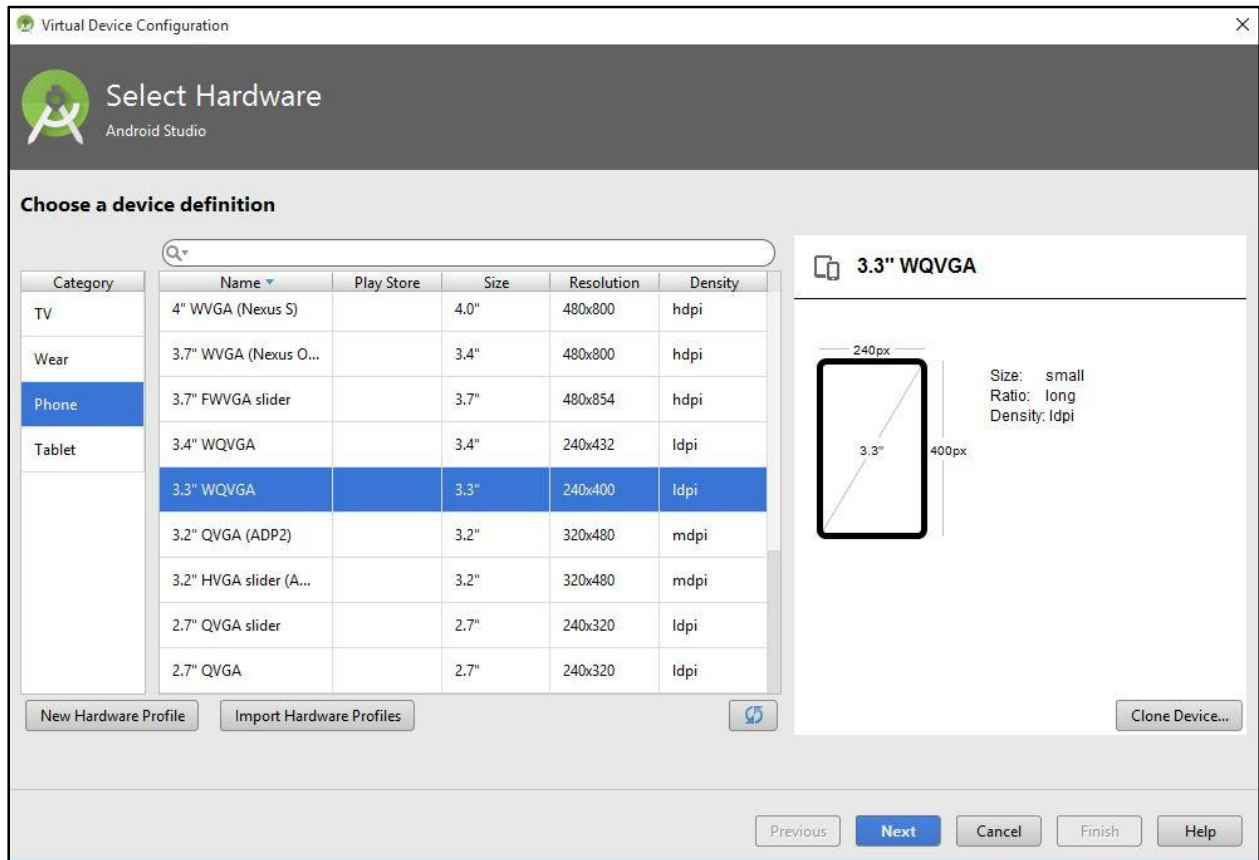


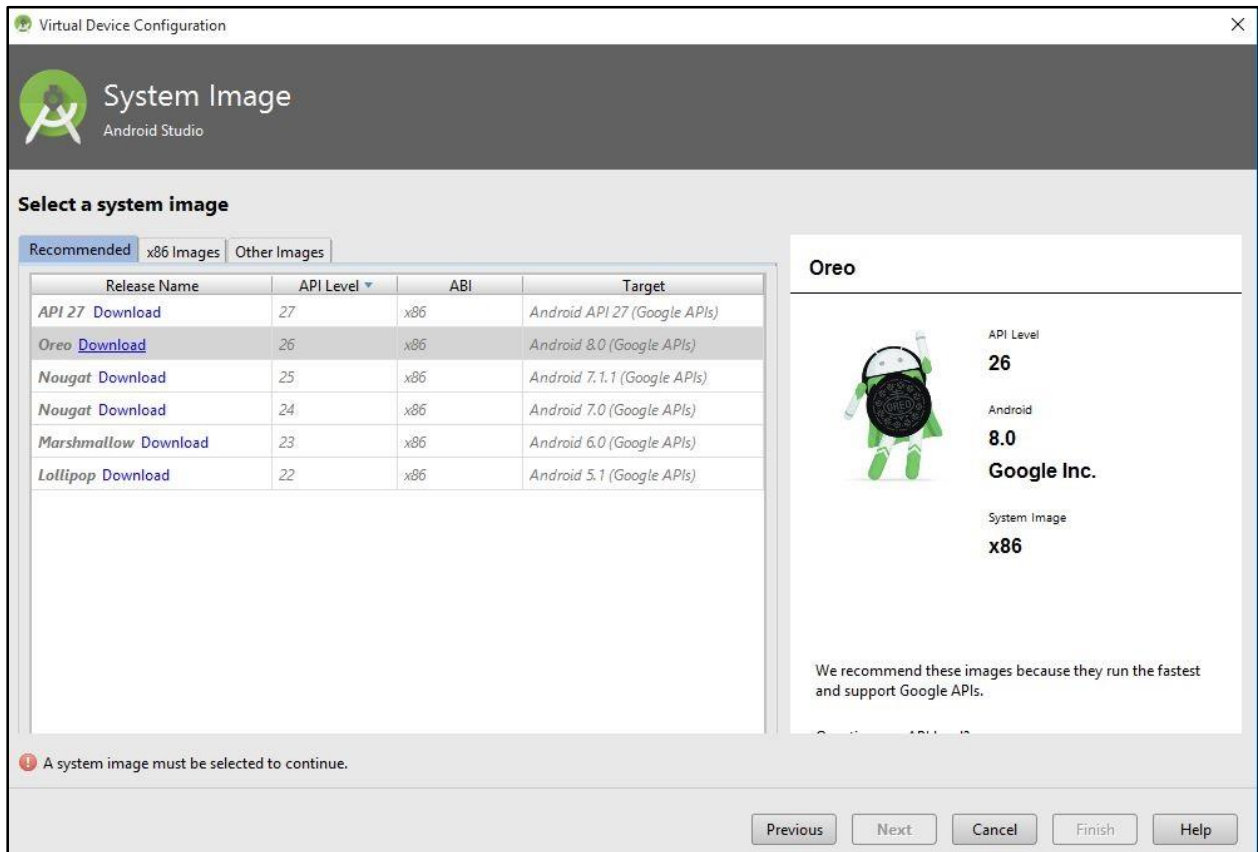
Figure 61 - Deployment Target selection

On the next window, you can select a phone you would like. In this example we will select the **3.3" WQVGA** Phone. Click **Next**.



**Figure 62 - Deployment Target selection**

On the Next Window, select the System image you would like to use. In our case, we will use the latest Android OS, Android 8 Oreo. Download this image



**Figure 63 - System Image selection**

Once it has downloaded, select the image, and then click **Next**.

On the next window, you can name your Virtual Device that you created. Let us name it **MyAVD** on **AVD Name**, as shown below:

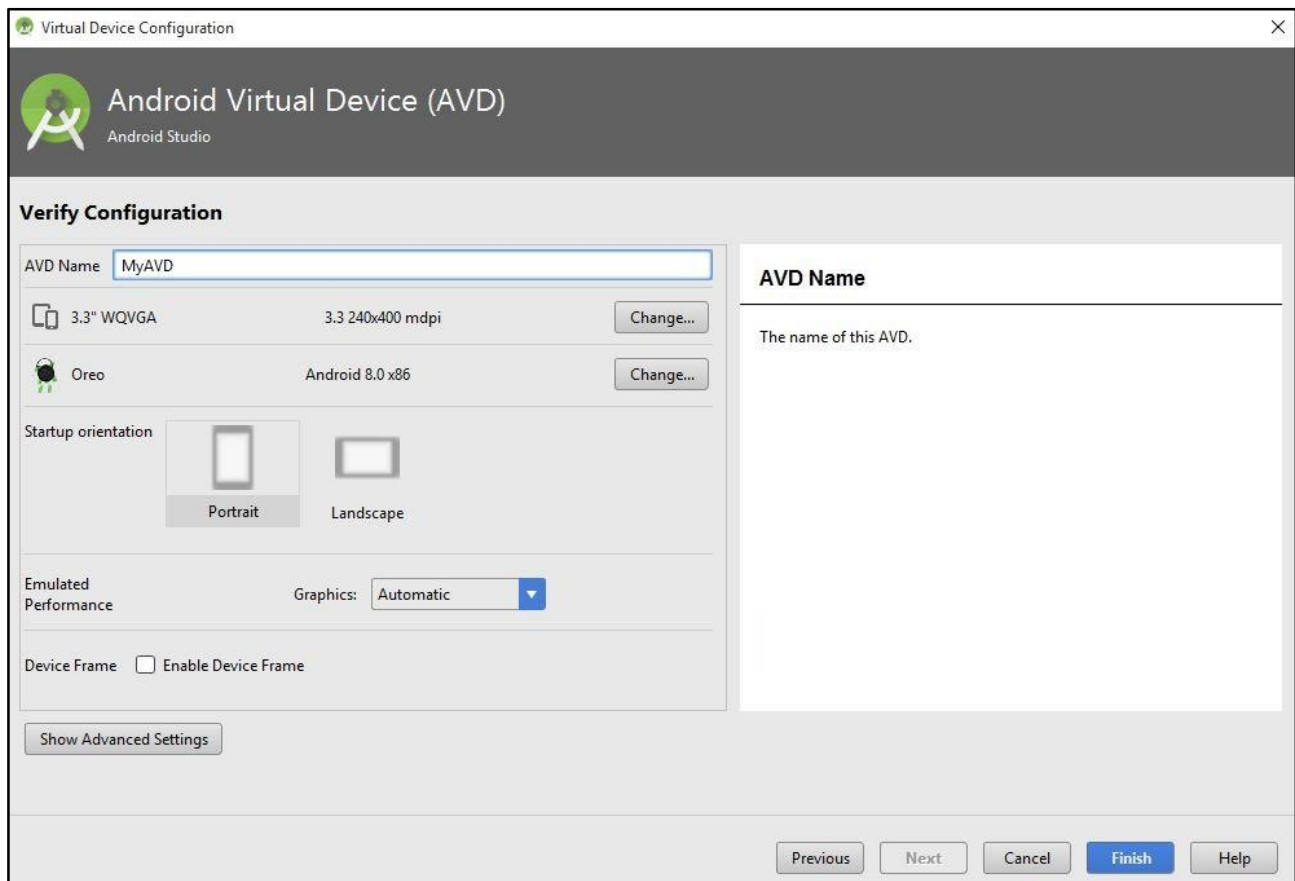


Figure 64 - Naming the Android Virtual Device

Click **Finish**.

**Note** If you receive a message saying “**HAXM is not installed**”, install this using the SDK Manager. Go to **Tools-> Android->SDK Manager**. Click SDK tools and find the HAXM package from list. HAXM (Hardware accelerated execution manager) is required to enable the emulator to run properly on your computer.

You will be returned to the Deployment Target Window, which will display the Virtual Device that we created. Click **OK**.

The Android Emulator will launch. Wait until it finishes. This process could take time, depending on the speed of your computer. Once it has finished loading, our app will be displayed as below:

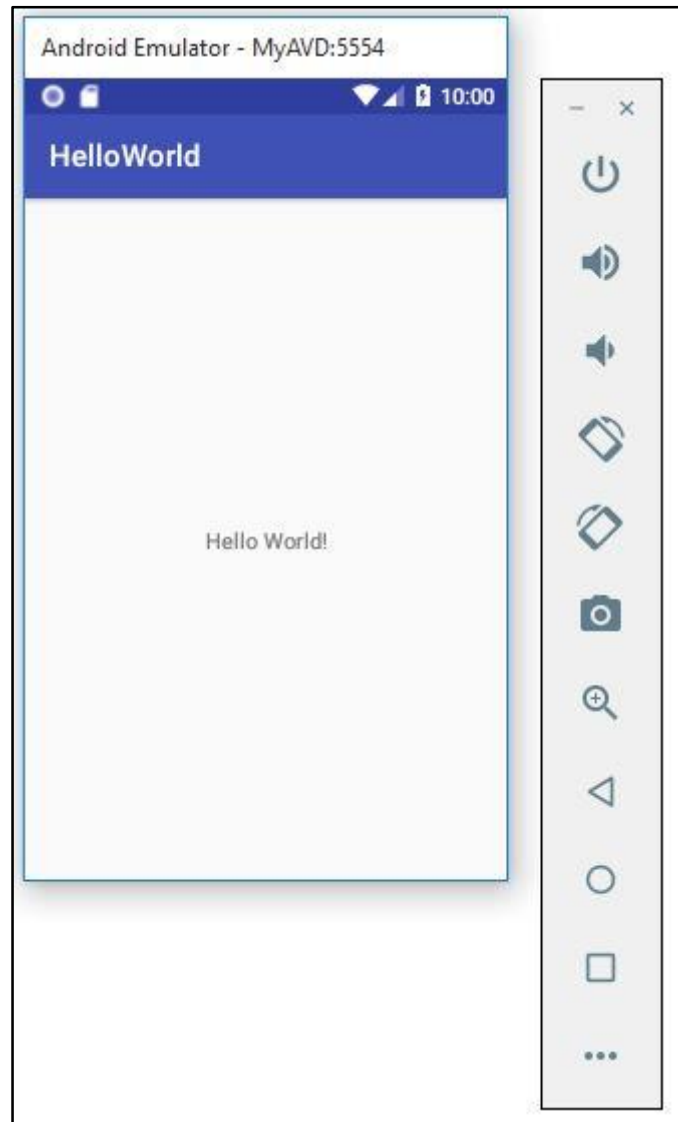


Figure 65 - HelloWorld app running on emulator

Congratulations, you have created your first Android app.

### 1.13 Building user interfaces

The basic building block for user interfaces is the **view** object, which is created from the **View** class; such occupies a rectangular area on screen and is responsible for drawing and event handling. **View** is the base class for widgets, which are used to create interactive user interface components, such as buttons, text fields, etc. **viewGroup** is a subclass of **View**; such provides an invisible container that holds other **views** or **viewGroups** and define their layout properties.

There are different layouts, which are subclasses of the **ViewGroup** class; a typical layout defines the visual structure of an Android user interface and can be created either at runtime

using **View/ViewGroup** or you can declare your layout using the simple XML file **main\_layout.xml**, which is located in the **res/layout** folder of your project.

A layout may contain any type of widget, such as buttons, labels, text boxes, etc. Figure 66 is a simple example of an XML file having **LinearLayout**:

```
<?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:id = "@+id/text"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "This is a TextView"/>

<Button
    android:id = "@+id/button"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "This is a Button"/>

    <!-- More GUI components go here -->
</LinearLayout>
```

**Figure 66 – LinearLayout**

A **view** object may have a unique **ID** assigned to it, which will identify **View** uniquely within the tree. The syntax for an **ID** inside an XML tag is:

```
android:id = "@+id/my_button"
```

The following is a brief description of the **@** and **+** signs:

- The at-symbol (**@**) at the beginning of a string indicates that the XML parser should parse and expand the rest of the **ID** string and identify such as an **ID** resource.



- The plus-symbol (+) means that this is a new resource name that must be created and added to resources. To create an instance of the **view** object and capture it from layout, use the following:

```
Button myButton = (Button) findViewById(R.id.my_button);
```

Once your layout is defined, you can load the layout resource from your application code in your `Activity.onCreate()` callback implementation, as shown below (Figure 60):

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
}
```

Figure 67 – How to load layout resources from application code

## 1.14 User interface controls

An Android application user interface is everything that a user can see and interact with. You have learnt about the various layouts that you can use to position your views in an activity. This section will provide you with more detail on the various views.

A **view** is an object that draws something on a screen that a user can interact with whereas a **viewGroup** is an object that holds other **view** (and **viewGroup**) objects in order to define the layout of the user interface. You define layout in an XML file, which offers a human-readable structure for layout, similar to HTML. For example, a simple vertical layout with a **textView** and **button** will look like this (Figure 68):

```
<?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:id = "@+id/text"  
        android:layout_width = "wrap_content"  
        android:layout_height = "wrap_content"  
        android:text = "I am a TextView"/>
```

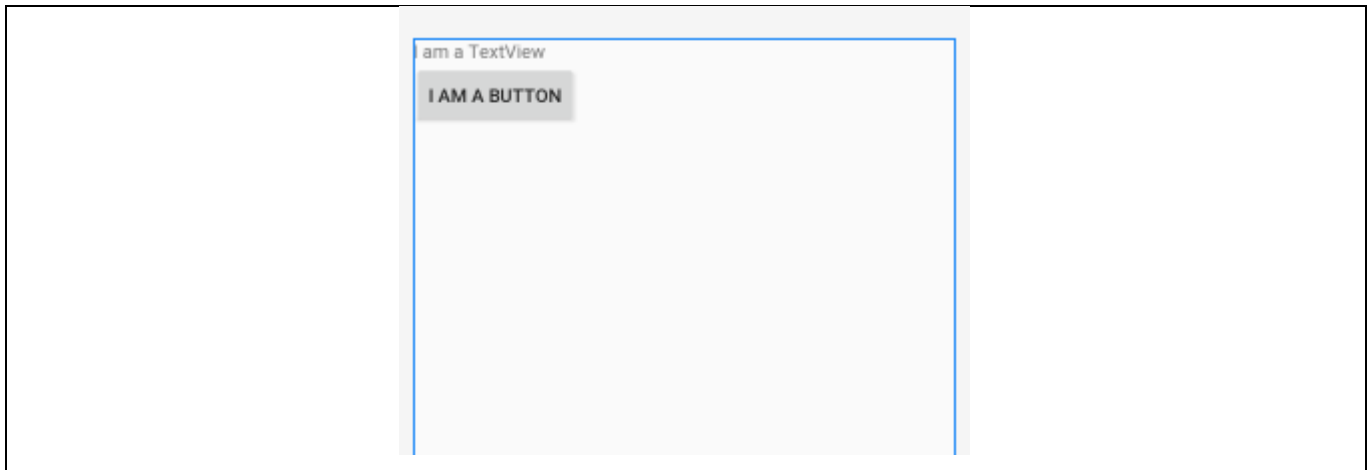
```

<Button android:id = "@+id/button"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:text = "I am a Button"/>
</LinearLayout>

```

**Figure 68 – Simple vertical layouts with textView**

When viewing the layout, it should display the widgets as below:



**Figure 69 – TextView and Button displayed**

Table 4 summarises the user interface controls:

Number	User interface control	Description
1	textView	This view displays text to a user
2	editText	This view is a predefined subclass of <b>TextView</b> ; it includes rich editing capabilities
3	autoComplete TextView	This view is similar to <b>editText</b> , except it shows a list of completion suggestions automatically while a user is typing
4	button	A push-button that can be pressed or clicked by a user to perform an action
5	imageButton	This view enables you to specify the exact location of children

6	checkBox	An on/off switch that can be toggled by a user; use such when presenting users with a group of selectable options that are <b>not</b> mutually exclusive
7	toggleButton	An on/off button with a light indicator
8	radioButton	This view has <b>two</b> states, namely, checked or unchecked
9	radioGroup	This view groups <b>one</b> or more <b>radioButtons</b> together
10	progressBar	This view provides visual feedback about ongoing tasks, such as when a task is being performed in the background
11	spinner	A drop-down list that allows a user to select <b>one</b> value from a set
12	timePicker	This view enables a user to select a time of day in either 24-hour mode or AM/PM mode
13	datePicker	This view enables a user to select a date

**Table 4 – User interface controls**

Source: [http://www.tutorialspoint.com/android/android\\_tutorial.pdf](http://www.tutorialspoint.com/android/android_tutorial.pdf)

### 1.15 Creating user interface controls

As explained previously, a **view** object may have a unique **ID** assigned to it, which will identify such uniquely within a tree. The syntax for an **ID** inside an XML tag is:

`android:id = "@+id/text_id"`

To create a user interface control/view/widget, you will have to define a view/widget in the **Layout** file and assign such a unique **ID** (Figure 70):

```
<?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">
```

**Figure 70 – How to create views**

Finally create an instance of the **control** object and capture such from the layout using the following:

```
TextView myText = (TextView) findViewById(R.id.text_id);
```

## 1.16 Event listeners

Events are a useful way to collect data about user interaction with interactive components, such as button presses, screen touch, etc. The Android Programming Framework maintains an event queue into which events are placed as they occur; each event is removed from said queue on a FIFO basis. You can capture such events in your program and then take appropriate action as per your requirements.

The following **three** concepts are related to Android event management:

### 1. Event listeners:

The **View** class is mainly involved in building up Android GUIs and providing a number of event listeners. An **eventListener** is an object that receives notification when an event happens.

### 2. Event listener registration:

Event registration is a process wherein an event handler gets registered with an event listener so that said handler is called when said listener fires an event.

### 3. Event handlers:

When an event happens and you have registered an event listener for such, said listener calls **eventHandler**, which is a method that actually handles the event.

Table 5 summarises event listeners and handlers:

Event handler	Event listener	Description
<code>onClick()</code>	<code>onClickListener()</code>	This is called when a user clicks, touches or focuses on any widget, such as a button, text, image, etc
<code>onLongClick()</code>	<code>onLongClickListener()</code>	This is called when a user clicks, touches or focuses on any widget, such as a button, text, image, etc. for <b>one</b> or more seconds

<code>onFocusChange()</code>	<code>onFocusChangeListener()</code>	This is called when a widget loses its focus, i.e. when a user goes away from the <b>view</b> item
<code>onKey()</code>	<code>onKeyListener()</code>	This is called when a user is focused on an item and presses or releases a hardware key on a device
<code>onTouch()</code>	<code>onTouchListener()</code>	This is called when a user presses a key, releases a key or any other movement gesture on a screen
<code>onMenuItemClick()</code>	<code>onMenuItemClickListener()</code>	This is called when a user selects a menu item

**Table 5- Event Handlers and Listeners**

### 1.16.1 Event listeners registration

Event registration is a process wherein an event handler gets registered with an event listener so that said handler is called when said listener fires an event. There are several tricky ways to register event listeners, however, we are only going to list the top **three** ways out of which you can choose, based on your situation:

1. Using the anonymous **Inner** class
2. Using the **Activity** class to implement the **Listener** Interface
3. Using the layout file **activity\_main.xml** to specify the event handler directly. The section that follows will provide you with event handling examples.

### 1.16.2 Event listeners registration using anonymous inner classes

Here, you will create an anonymous implementation of the event listener; such will be useful if each class is applied to a single control **only** as you will have the advantage of passing arguments to the event handler. In this approach, event handler methods can access the private data of **Activity**. No reference is needed to call **Activity**.

However, if you have applied the handler to more than **one** control, you would have to cut- and-paste the code for such; if said code is long, it makes such harder to maintain.

The following are simple steps showing how to make use of a separate **Listener** class to register and capture a **click** event. Similarly, you can implement a listener for any other required event type:

- Use Android Studio to create an Android application; name such **EventDemo** under, as explained in the “**Hello World**” example.
- Modify the **src/MainActivity.java** file to add **click** event listeners and handlers for the **two** buttons as defined.
- Modify the default content of the **res/layout/activity\_main.xml** file to include Android user interface controls.
- Define the required constants in the **res/values/strings.xml** file.
- Run the application to launch the Android Emulator and verify the result of the changes as made to the application.

Figure 71 is the content of the modified main activity file

**src/com.example.eventdemo/MainActivity.java**; this file can include each of the fundamental life cycle methods:

```
package com.example.eventdemo;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //--- find both the buttons ---
        Button sButton = (Button) findViewById(R.id.button_s);
        Button lButton = (Button) findViewById(R.id.button_l);
        //--- register click event with first button ---
        sButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
```

```

        //--- find the textView ---
        TextView txtView = (TextView) findViewById(R.id.text_id);
        //--- change text size --- txtView.setTextSize(14);
    }
});

//--- register click event with second button ---
lButton.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        //--- find the textView ---
        TextView txtView = (TextView) findViewById(R.id.text_id);
        //--- change text size ---
        txtView.setTextSize(24);
    }
});
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}

```

**Figure 71 – How to register event listeners**

Figure 72 is the content of the **res/layout/activity\_main.xml** file:

```

<?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">

    <Button
        android:id = "@+id/button_s"

```

```

        android:layout_height = "wrap_content"
        android:layout_width = "match_parent"
        android:text = "@string/button_small"/>

<Button
    android:id = "@+id/button_1"
    android:layout_height = "wrap_content"
    android:layout_width = "match_parent"
    android:text = "@string/button_large"/>

<TextView
    android:id = "@+id/text_id"
    android:layout_width = "wrap_content"
    android:layout_height = "wrap_content"
    android:capitalize = "characters"
    android:text = "@string/hello_world"/>
</LinearLayout>

```

Figure 72 – res/layout directory

Figure 73 is the content of **res/values/strings.xml** used to define **two** new constants:

```

<resources>
    <string name = "app_name">EventDemo</string>
    <string name = "action_settings">Settings</string>
    <string name = "hello_world">Hello World!</string>
    <string name = "button_small">Small font</string>
    <string name = "button_large">Large font</string>
</resources>

```

Figure 73 – Simple string directories

Figure 74 is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.guidemo">
    <uses-sdk

```



```

        android:minSdkVersion = "8"
        android:targetSdkVersion = "17"/>
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCH
ER" />
        </intent-filter>
    </activity>
</application>

</manifest>

```

Figure 74 – Android Manifest file

### 1.17 Intents

An Android **intent** is an object carrying intent, i.e. a message from **one** component to another within or outside of an application. **intent** can communicate messages among any of the **three** core components of an application, namely:

1. Activities
2. Services
3. Broadcast receivers

An **intent** object is a passive data structure that holds an abstract description of an operation to be performed. For example, assume that you have an **Activity** that needs to launch an e-mail client and send an e-mail using an Android device. For this purpose, your **Activity** would send an **ACTION\_SEND** along with the appropriate **chooser** to the Android **intent** resolver. The specified chooser provides the proper interface for the user to pick how to send e-mail data.

Table 6 summarises the mechanisms for delivering intent to each type of component:

Number	Intent method	Description
1	<code>context.startActivity()</code>	The <b>intent</b> object is passed to the method to launch a new activity or to get an existing activity to do something new
2	<code>context.startService()</code>	The <b>intent</b> object is passed to the method to initiate a service or to deliver new instructions to an ongoing service
3	<code>context.sendBroadcast()</code>	The <b>intent</b> object is passed to the method to deliver the message to all interested broadcast receivers

Table 6 – Intent

### 1.17.1 Intent objects

An **intent** object is a bundle of information, which is used by components that receive the **intent** as well as information used by the Android system. An **intent** object can contain the following components based on what it is communicating or going to perform:

- **Action**

This is a mandatory part of an **intent** object; it is a string that names the action to be performed or, in the case of broadcast intents, the action that took place and which is being reported on. **Action** largely determines how the rest of an **intent** object is structured. The **Intent** class defines a number of **action** constants corresponding to different **intent**. The **action** in an **intent** object can be set by the **setAction()** method and read by the **getAction()** method.

- **Data**

This encompasses the URI of the data that is to be acted upon and the MIME type of such data. For example, if the action field is **ACTION\_EDIT**, the data field would contain the URI of the document to be displayed for editing. The **setData()** method specifies data only as a URI, **setType()** specifies it only as a MIME type and **setDataAndType()** specifies it as both an URI and MIME. The URI is read by **getData()** and the type by **getType()**.

- **Category**

This is an optional part of an intent object; it is a string that contains additional information about the kind of component that should handle the intent. The **addCategory()** method places

a category in an intent object, `removeCategory()` deletes a category previously added and `getCategories()` gets the set of all categories currently in an object.

You need to understand how we use category to choose the appropriate activity to correspond with intent.

To implement intent, follow the following steps:

- Use Android Studio to create an Android application; name such **IntentDemo** under package **com.example.intentdemo**. While creating this project, ensure that you Target SDK and Compile With the latest version of Android SDK in order to use higher levels of API.
- Modify the `src/MainActivity.java` file and add code to define two listeners corresponding to two buttons, i.e. Start Browser and Start Phone.
- Modify the `res/layout/activity_main.xml` layout file to add three buttons to `LinearLayout`.
- Modify the `res/values/strings.xml` to define the required constant values.

Run the application to launch the Android Emulator and verify the result of the changes as made to the application.

Figure 75 is the content of the modified `src/com.example.intentdemo/MainActivity.java` file:

```
package com.example.intentdemo;
import android.net.Uri;
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.*;
import android.widget.Button;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);
        Button startBrowser = (Button) findViewById(R.id.start_browser);
        startBrowser.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
```

```

        Intent i = new Intent(android.content.Intent.ACTION_VIEW,
Uri.parse("http://www.example.com"));
        startActivity(i);
    }
});

Button startPhone = (Button) findViewById(R.id.start_phone);
startPhone.setOnClickListener(new View.OnClickListener() {
    public void onClick(View view) {
        Intent i = new Intent(android.content.Intent.ACTION_VIEW,
Uri.parse("tel:9510300000"));
        startActivity(i);
    }
});
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    //inflate the menu; this adds items to the action bar if it is
present
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}
}

```

**Figure 75 – How to modify Main Activity files**

Figure 76 is the content of the **res/layout/activity\_main.xml** file:

```

<LinearLayout xmlns:android =
"http://schemas.android.com/apk/res/android"
    android:layout_width = "fill_parent"
    android:layout_height = "fill_parent"
    android:orientation = "vertical">

    <Button
        android:id = "@+id/start_browser"

```

```

        android:layout_width = "fill_parent"
        android:layout_height = "wrap_content"
        android:text = "@string/start_browser"/>

<Button
    android:id = "@+id/start_phone"
    android:layout_width = "fill_parent"
    android:layout_height = "wrap_content"
    android:text = "@string/start_phone"/>
</LinearLayout>

```

**Figure 76 – res/layout file**

Figure 77 is the content of the res/values/strings.xml file used to define the two new constants:

```

<resources>
    <string name = "app_name">IntentDemo</string>
    <string name = "action_settings">Settings</string>
    <string name = "hello_world">Hello World!</string>
    <string name = "start_browser">Start browser</string>
    <string name = "start_phone">Start phone</string>
</resources>

```

**Figure 77– String file**

Figure 78 is the default content of **AndroidManifest.xml**:

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.intentdemo">
    <uses-sdk
        android:minSdkVersion = "8"
        android:targetSdkVersion = "17"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"

```

```

        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCH
ER" />
            </intent-filter>
        </activity>
    </application>

</manifest>

```

**Figure 78 – Manifest file with Intent declaration**

Run your application and observe the results.

## 1.18 Android Studio and SQLite Databases

Android Studio gives a functionality to use SQLite databases in the event you want to save repeating or structured data. The APIs you'll need to use a database on Android are available in the `android.database.sqlite` package.

There are two main classes in this package, namely the `SQLiteOpenHelper` and the `SQLiteCursor` classes. The `SQLiteOpenHelper` class is helper class to manage database creation and version management. The `SQLiteCursor` class exposes results from a query on a `SQLiteDatabase`.

To have a better understanding on how to implement these classes in Android Studio, follow this tutorial on this link: <https://www.journaldev.com/9438/android-sqlite-database-example-tutorial>

. It will show you how to develop an Android application that performs CRUD operations.

This tutorial will prove useful as you prepare for your project.

In addition the following YouTube tutorial series on how you can develop apps that communicate with SQLite databases :

<https://www.youtube.com/playlist?list=PLrnPJCHvNZuAPyh6nRXsvf5hF48SJWdJb> .

### 1.18.1 Connecting Android Studio to XAMPP

It is possible that you would want to develop an application that communicates to an external database like one hosted on a XAMPP server. With Android Studio, you are able to do this.

Follow the YouTube tutorial link here to see how it can be done:

<https://www.youtube.com/watch?v=ryMj8xnZkSQ>

This will prove useful as you prepare for your project.

### **Concluding remarks**

This unit discussed how to build applications using an Android Studio. The main Android components were explained as well as how such can be integrated. We also explored how to build user interfaces with in Android Studio. Implementation of Intents was also explained. The unit finished by providing links to YouTube tutorials on how one can develop Android apps that make use of SQLite Databases as well as an external database such as XAMPP.