DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A MINI PROJECT REPORT

ON

HIKER'S WATCH, AN APP FOR THE ADVENTUROUS

Submitted in partial fulfillment

for the award of Bachelor of Engineering

In

COMPUTER SCIENCE AND ENGINEERING

Submitted by

SAMRIDHI GUPTA
1NH18CS168
4 C

DURING

EVEN SEMESTER 2019-2020

For

COURSE CODE

19CSE48

Reviewed by

Ms. DEEPTI RAI

(ASST. PROFESSOR, DEPT. OF CSE)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is to certify that the mini project work titled

HIKER'S WATCH, AN APP FOR THE ADVENTUROUS

Submitted in partial fulfillment of the degree of Bachelor of Engineering

SAMRIDHI GUPTA

1NH18CS168

DURING

EVEN SEMESTER 2019-2020

For

19CSE48

Signature of Reviewer

Signature of HOD

SEMESTER END EXAMINATION

Name of the Examiner	*Signature with date*
1._____	2._____
_____	_____

ABSTRACT

Mobile app development is the act or process by which a mobile app is developed for mobile devices, such as personal digital assistants, enterprise digital assistants or mobile phones. These applications can be pre-installed on phones during manufacturing platforms, or delivered as web applications using server-side or client-side processing (e.g., JavaScript) to provide an "application-like" experience within a Web browser. Application software developers also must consider a long array of screen sizes, hardware specifications, and configurations because of intense competition in mobile software and changes within each of the platforms. Mobile app development has been steadily growing, in revenues and jobs created. A 2013 analyst report estimates there are 529,000 direct *app economy* jobs within the EU then 28 members (including the UK), 60 percent of which are mobile app developers.

In this project we are going to make an app which helps us to find the Geostationary location of any given place at any given time. “Hikers Watch” basically implements the usage of functions like geocoder , java OOP, inheritance and various other functions. The app is built on the Android Studio platform because its user friendly and one of the best environment to create and run any application for an android developer. This app supports all android devices and uses reverse geocoder technique to gain the longitudinal and latitude of an user at any given time.

KEYWORDS

Geocoder

Reverse Geocoder

Android Studio

Java OOP

Inheritance

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

I have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman of New Horizon Educational Institutions for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha,** Principal NHCE, for his constant support and encouragement.

I am grateful to **Dr. Prashanth C.S.R**, Dean Academics, for his unfailing encouragement and suggestions, given to me in the course of my project work.

I would also like to thank **Dr. B. Rajalakshmi**, Professor and Head, Department of Computer Science and Engineering, for her constant support.

I express my gratitude to **Ms. Deepti Rai,** Asst. professor CSE department, my project guide, for constantly monitoring the development of the project and setting up precise deadlines. Her valuable suggestions were the motivating factors in completing the work.

SAMRIDHI GUPTA (1NH18CS168)

CONTENTS

LIST OF FIGURES

NAME : SAMRIDHI GUPTA

USN : 1NH18CS168

TITLE : HIKER'S WATCH, AN APP FOR THE ADVENTUROUS

SEM : 4TH

SECTION : C

CHAPTER 1

INTRODUCTION

1.1 APP DEVELOPMENT AND IT'S SCOPE

India stands second for internet users after China and is expected to grow in 2020. With the increase in market shares for Android in India, most of the companies are now getting android app development. Moreover, Android is a free source. It is the fastest growing market in India.

Android Studio provides a unified environment where you can build apps for Android phones, tablets, Android Wear, Android TV, and Android Auto. Structured code modules allow you to divide your project into units of functionality that you can independently build, test, and debug.

Android Studio includes project and code templates that make it easy to add well-established patterns such as a navigation drawer and view pager.

1.2 OBJECTIVES

Android development can be comfortably done in either Java or Kotlin. The objective of this project is to create a simulation of the Hiker's Watch, an app for the adventurous. The code would be purely written in Java. The aim is to provide the user with:-

- A one click working simulation
- Application of the google maps and user location
- Altitude of the demanded place
- Latitude and the longitude of the demanded place
- Address of the place that is entered

1.3 METHODOLOGY TO BE FOLLOWED

Android Studio is the official integrated development environment (IDE) for Google's android operating system, built on JetBrains' Intellij's IDEA software and designed specifically for Android Development. The whole project would be developed in the Android Studio and for using the maps, we would require the permission to use Google APIs.

A Project in Android Studio contains everything that defines a workspace for your app, and provides you with packages and build configurations. On the start of a new project, Android Studio creates the necessary structure for all the files. The key components for your project are provided.

The project gets divided into various, discrete units of functionality called the modules. Each module can be tested, built and debugged independently. They can also be dependent on each other.

In the **Create New Module** window, Android Studio offers the following types of app modules:

1. Phone & Tablet Module
2. Wear OS Module
3. Android TV Module
4. Glass Module

The Hiker's watch would require the help of the Google Application Interfaces, to access the user's location on the device. **Google APIs** is a set of application programming interfaces developed by google which allow communication with Google services and their integration to other services. Examples of these include Search, Gmail, Translate or Google Maps. Third-party apps can use these APIs to take advantage of or extend the functionality of the existing services.

So the first thing we do is to create a new project, and give our application a name. Here, my application has been named as the Hiker's watch. Next, we select an Activity which is like the design part of our app. In this project, I have chosen the empty activity. There are many activities provided like the basic activity, Navigation activity, the Map activity, etc.

Once the project is created, we go to the ActivityManifest.xml part of the workspace. Here, we decide how our app is going to look, i.e. whether we choose it to be a full screen app or not. My project implements the usage of the full screen app look.

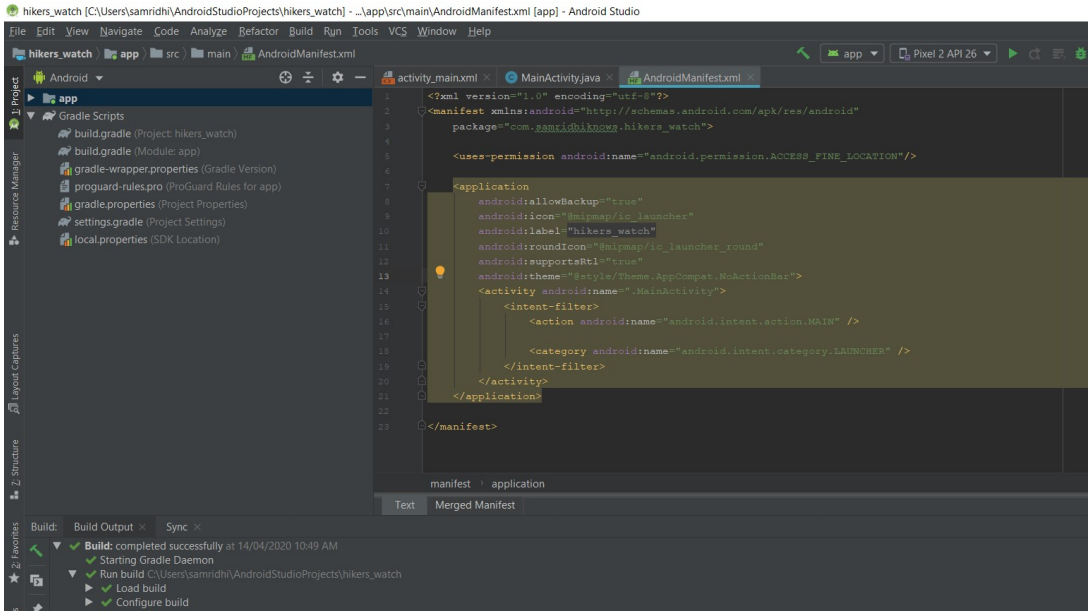

Fig 1: The screenshot displays the AndroidManifest.xml window

The complete project is coded in the MainActivity.java part of the workspace. The various concepts that have been implemented in the project, have been looked into detail in the below sections. The most important feature being implemented in the app is Reverse Geocoding. This means that, when I set any location from around the world on the map, let's say **New Horizon College of Engineering**, the Hiker's Watch app would display the latitude, longitude, accuracy and the altitude of the college i.e. the latitude of the college is 12.93372 and longitude of the college is 77.6927. If the app isn't able to find the address of the location or if the location is unknown, it displays an error message telling 'could not find location'.

One of the most essential part of app designing is the outlook of the app. The various features of any app that you see includes texts, images, audios and toasts (toasts are pop up messages that you get while you run the app or get an output). The designing of the hiker's watch takes place in the activity_main.xml section of the workspace. Here, I have included many texts views and the background image, which serves as the image view. Among the text views are the longitude, latitude, altitude, accuracy and the address of the user's location.

A suitable emulator is chosen. The emulator I have chosen for my app is the Pixel 2 API 26. Once the run command is given, the emulator displays the app (if the code has no errors and the configurations have been built successfully!).

Setting up the AVD or the Android Virtual Device is also essential. It determines how the app is going to look in the phone, tv, watch OS or any other mode. In this project, I have used the Pixel 2 API 26 AVD, which gives my app a smartphone look. That's where the output of the app is shown. The AVD implements the Oreo 8.1 version of android, which was the eight major release and the 15th version of the Android app system.

1.4 OUTCOMES

The simulation of the Watch would appear on the screen at a single click of the run button. A device which looks like your smartphone, would appear.

Fig 2. An outlook of the app

1.5 HARDWARE AND SOFTWARE REQUIREMENTS

- Android Studio APK for app development
- LAPTOP-2TUG730U
- Installed RAM- 16.0 GB
- Processor – Intel core i7, 8th gen-8750H

CHAPTER 2

OBJECT ORIENTED CONCEPTS

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts −

- Polymorphism
- Multithreading
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Exception Handling
- this and super keywords
- Overriding
- Casting

2.1 THE ONCREATE METHOD AND ABSTRACTION

To navigate between the stages of the Activity throughout the running of the code, the class Activity provides us with some methods to keep a check on each activity or the status of the program, onCreate(), onResume(), onStart(), onStop(), onPause(), etc.

As the user begins to leave the activity, the system calls methods to dismantle the activity. In some cases, this dismantlement is only partial; the activity still resides in memory (such as when the user switches to another app), and can still come back to the foreground. If the user returns to that activity, the activity resumes from where the user left off. With a few exceptions, apps are restricted from starting activities when running in the background.

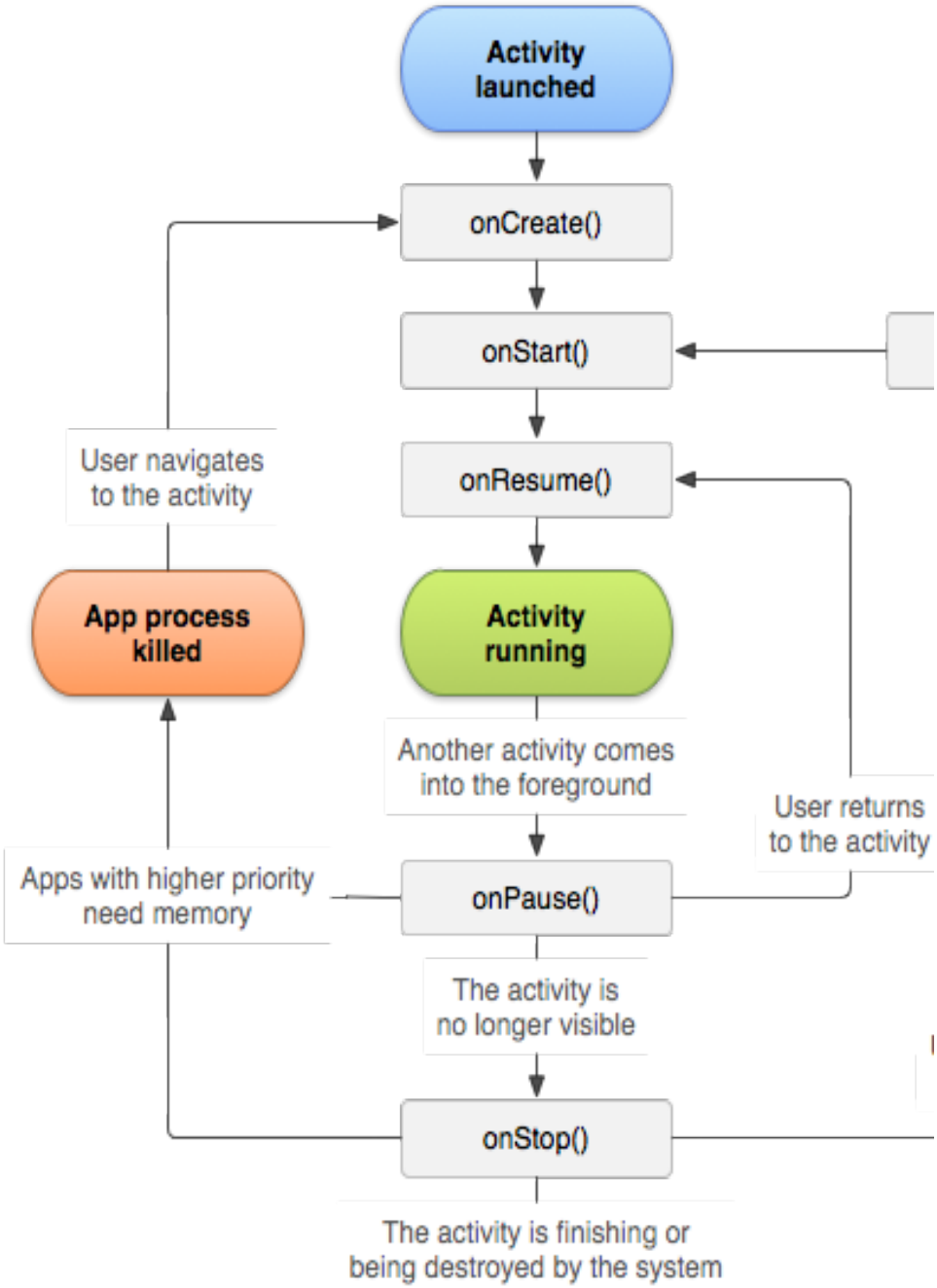

Fig 3. The working of onCreate method

You must implement this callback, which fires when the system first creates the activity. On activity creation, the activity enters the *Created* state. In the `onCreate()` method, you perform basic application startup logic that should happen only once for the entire life of the activity.

2.2 CLASS

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type and is declared by the class keyword. The data or variables within a class are known as instance variables.

Android Studio provides file templates that determine how new Java classes and types are created with the **Create New Class** dialog. You can customize these templates.

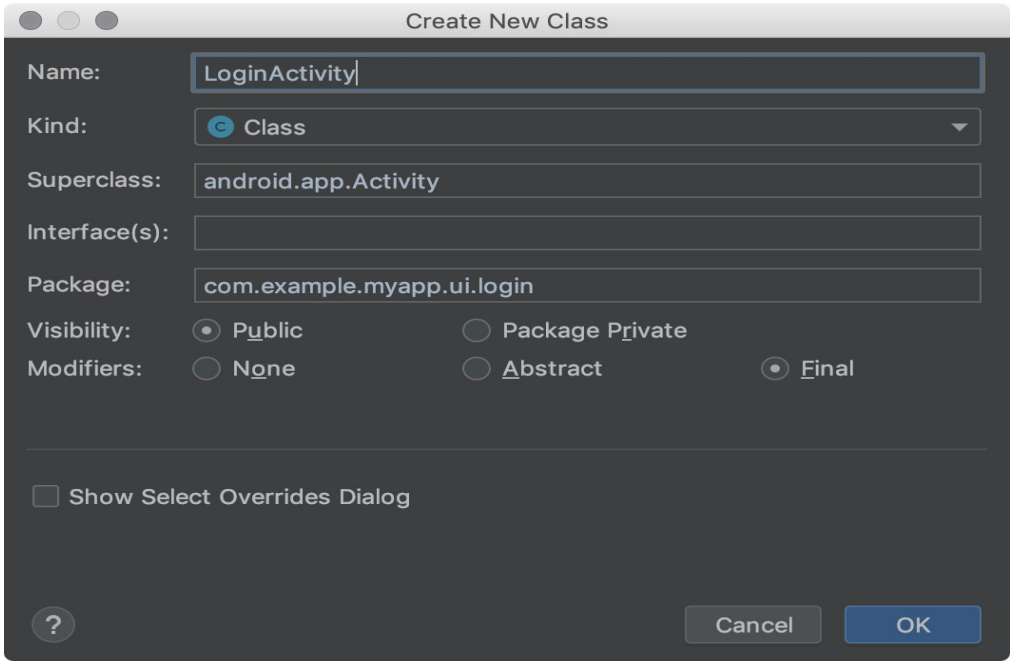

Fig 4. The creation of class in Android Studio

2.3 INHERITANCE

Inheritance is one of the cornerstones of object oriented programming, because it allows you to create hierarchical classifications. Using inheritance, we cab create a greneral class that defines traitis common to a set of related items. This class can then be later inherited by further, more classes. The class that is inherited is known as the superclasss and the class which does the inheritation, is known as the subclass.

They both have a child-parent like relationship. To inherit a class, we simply use the extends keyword.

The class **MainActivity** will be extending the class **AppCompatActivity.** This class would serve the base class for newer activities that we wish to incorporate. By this, the project would be implementing the concept of inheritance. The heirarchy of the AppcompatActivity can be defines as:

androidx.activity.ComponentActivity
↳ androidx.fragment.app.FragmentActivity
↳ androidx.appcompat.app.AppCompatActivity

```
public class MainActivity extends AppCompatActivity
```

Fig 5 inheritance of the AppCompatActivity class

2.4 METHODS

Method provides information about, and access to, a single method on a class or interface. The reflected method may be a class method or an instance method (including an abstract method). Some of the android studio methods are:

- equals()
- getAnnotations()
- getDefaultvalue()
- getExceptiontypes()
- hashcode()
- invoke()

There would be two really important methods which we would be needing for the building of the app. They are:-

1. The **LocationListener** method is used for receiving notifications from the LocationManager when the location has changed. These methods are called if the LocationListener has been registered with the location manager.
2. The LocationManager method class provides access to the system location services. These services allow applications to obtain periodic updates of the device's geographical location, or to be notified when the device enters the proximity of a given geographical location.

2.5 SUPER KEYWORD

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable. Here, the super keyword would be implemented by the onCreate() method to know the **savedInstanceState** of the activity.

```
@Override
public void onCreate(Bundle savedInstanceState) {
// call the super class onCreate to complete the creation of activity like
// the view hierarchy
super.onCreate(savedInstanceState);
```

The savedInstanceState is a reference to a Bundle object that is passed into the onCreate method of every Android Activity.

```
public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);

    if (grantResults.length>0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
        startListening();
```

Fig 6 Implementation of the super keyword

2.6 OVERRIDING

If subclass has the same method as declared in the parent class, it is known as **method overriding in Java**. In this project, the methods defined under the MainActivity class have implemented the concept of Overriding.

- public void onLocationChanged(Location location)

Called when the user wants to change the location of the place. For example, I want the address of New York, but the current location displayed is of Sydney. So this method would help in identifying the new location the user has entered and change the address on the app accordingly.

- public void onProviderEnabled(String provider)

This method would be called when the location provider is enables. This value must never be null.

- public void onProviderDisabled(String provider)

This method is called when the provider location is disabled and the app dosen't have access to the user's location.

- public void onStatusChanged(String provider, int status)

```
1.
2.
3.
4.    @Override
    public void onLocationChanged(Location location) {
        Log.i("Location", location.toString());
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {

    }

    @Override
    public void onProviderEnabled(String provider) {

    }

    @Override
    public void onProviderDisabled(String provider)
5.
```

Fig 7 Methods

2.7 EXCEPTION HANDLING

The exceptions handled in the code are with the help of the try and the catch block.

An exception is an abnormality that arises during the run time in the code. Other programming languages that do not support exception handling, have to be checked for errors manually. Java's exception handling avoids these problems.

Exceptions should be thrown in exceptional circumstances where the calling code needs to decide how to recover from an error condition.

```
try{
    List<Address>  listAddresses =
geocoder.getFromLocation(location.getLatitude(), location.getLongitude(), 1);

    if (listAddresses != null && listAddresses.size() > 0) {
        address = "Address: \n";

        if(listAddresses.get(0).getThoroughfare() != null) {
            address += listAddresses.get(0).getThoroughfare() + "\n";
        }

        if(listAddresses.get(0).getLocality() != null) {
            address += listAddresses.get(0).getLocality() + " ";
        }

        if(listAddresses.get(0).getPostalCode() != null) {
            address += listAddresses.get(0).getPostalCode() + " ";
        }

        if(listAddresses.get(0).getAdminArea() != null) {
            address += listAddresses.get(0).getAdminArea();
        }
    }
}catch(Exception e){
    e.printStackTrace();
```

Fig 8 Demonstration of the Exception Handling

Here, the geocoder has been put in the try catch block. If there is a problem in getting the locality, thorough fare, postal code and the admin area of the user's location, an exception would be thrown by the code, which would then be caught by the catch block. On the catching of the exception, a message is displayed on the app showing "Address not found :(".

Geocoding is the process of transforming a street address or other description of a location into a (latitude, longitude) coordinate. Reverse geocoding is the process of transforming a (latitude, longitude) coordinate into an address. The amount of detail in the reverse geocoded location description will vary, for example one might contain the full street address of the closest building, while another might contain only a city name and postal code. The geocoder is a class which handles geocoding.

To guard against the run time errors, simply enclose the code that you want to monitor inside the try catch block. Immediately after the try block, comes the catch block.

2.8 MULTITHREADING

The app implements multithreading implicitly as different tasks concurrently take place at the same time.The methods provided by the class Activity and the demonstration of the onCreate() method, implements multithreading. The geocoder method and the main method, are running at the same time. This can be seen in the 'Run' window, where the gradle build shows a message telling that 'the app might be doing too much work on the main thread'. Main threading splits one program into separate tasks, all interconnecting each other and working at the same time.

2.8 PACKAGES

```
package com.samridhiknows.hikers_watch;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import java.util.List;
import java.util.Locale;
```

Fig 9 Importing Packages

Package objects contain version information about the implementation and specification of a Java package. This versioning information is retrieved and made available by the ClassLoader instance that loaded the class. Typically, it is stored in the manifest that is distributed with the classes.

CHAPTER 3

DESIGN

3.1 DESIGN GOALS

The "main" activity starts when the user taps your app's icon. You can also direct the user to an activity from elsewhere, such as from a notification or even from a different app.

Android allows you to provide different resources for different devices. For example, you can create different layouts for different screen sizes. The system determines which layout to use based on the screen size of the current device.

If any of your app's features need specific hardware, such as a camera, you can query at runtime whether the device has that hardware or not, and then disable the corresponding features if it doesn't. You can specify that your app requires certain hardware so that Google Play won't allow the app to be installed on devices without them.

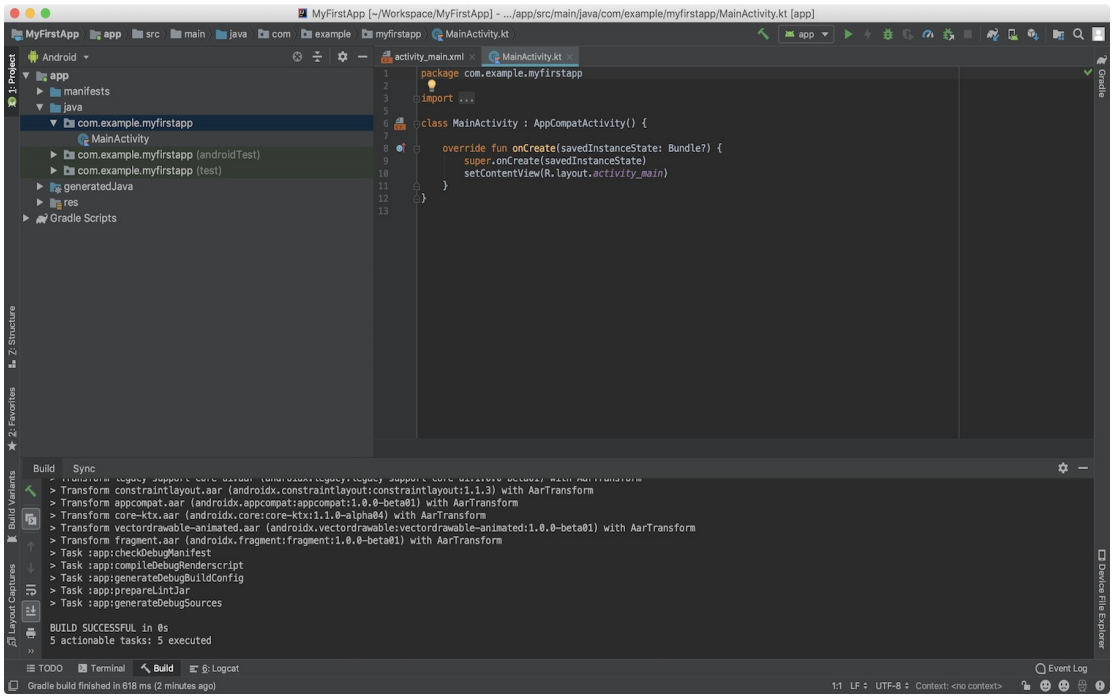

Fig 10 The Main activity window

We can design the outlook of our app in the activity_main.xml. It acts like the GUI. This page provides us with several buttons, which comprise of the text buttons, image buttons, several layouts, provision to add audios and videos to our app. To add an audio visual component to our app, we need to download them in the res file first. I have used an image of a camping night in the background. This would make my app more attractive. The text and the image buttons can be dragged into the constraint layout. The text buttons used in this app are the latitude, longitude, altitude, accuracy and the address of the user's location.

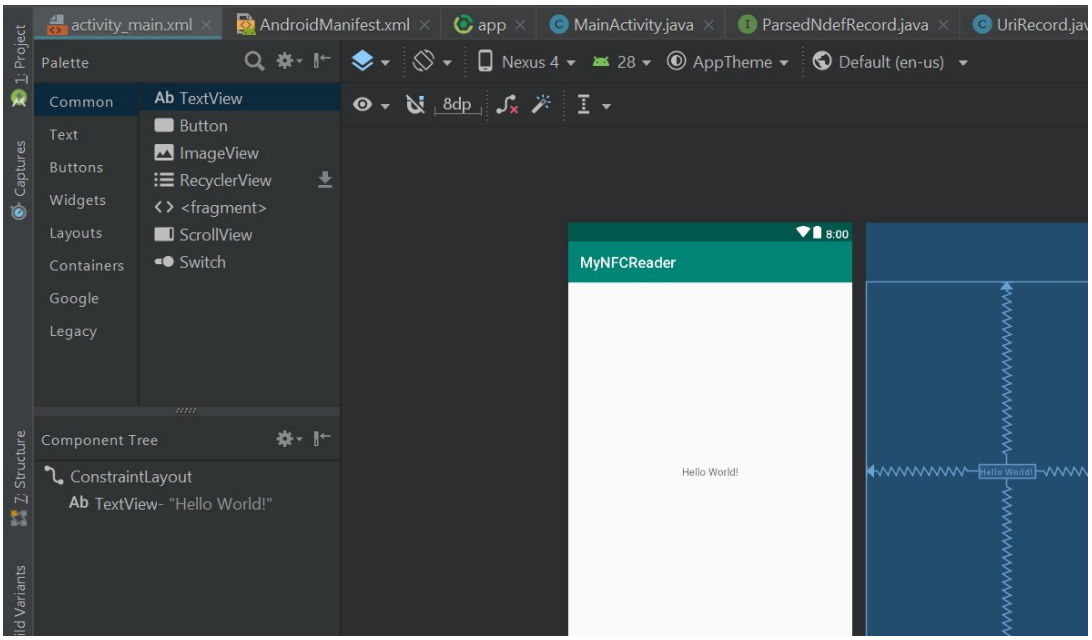

Fig 11 The window for designing the app

3.2 ALGORITHM

- Import all the packages
- Set up the locationManager and locationListener. This implements casting and overriding.
- Check whether the user has provided an access to the location. The google API for maps is set up in the manifest window.
- If access is not provided, we need to request for the location using the 'request Permissions' method.
- Once we get the permission, we need to get the location updates and pass the GPS_PROVIDER.
- Next, we need to get the last known location.
- The Android Manifest file (AndroidManifest.xml) of your app, indicate that your app is standalone. Specifically, add a meta-data
- Once we are set up with this, we can go ahead and update the location information. The updateLocationInfo method does this task.
- We need to get access to all the text views which are the latitude, longitude, altitude and accuracy of the user's location. For this, we create objects of the reference type updateLocationInfo method.
- Next, we define the geocoder method.
- The whole block is put into try and catch clauses, with multiple try statements and one catch statement to catch the exception. A warning message is displayed if any exception occurs.

CHAPTER 4

IMPLEMENTATION

4.1 OUTLOOK OF THE APP

In android, **TextView** is a user interface control that is used to set and display the text to the user based on our requirements. The TextView control will act as like label control and it won't allow users to edit the text.

The first step would be to display the title of the app i.e. the Hiker's Watch. So, the first button that would be used is the text button. The id of the button can be changed to 'Hiker's watch' and likewise the text message to be displayed can be written into it. The constraints can be changed according to your convenience. By making the suitable alignments of the button and giving the text we require, we can paste rest of the buttons. After this, comes the latitude and the longitude buttons. Likewise, the rest of the buttons can be tweaked according to the text messages you want and can be pasted on the image you have setup or on the background. This completes the outlook of the user screen.

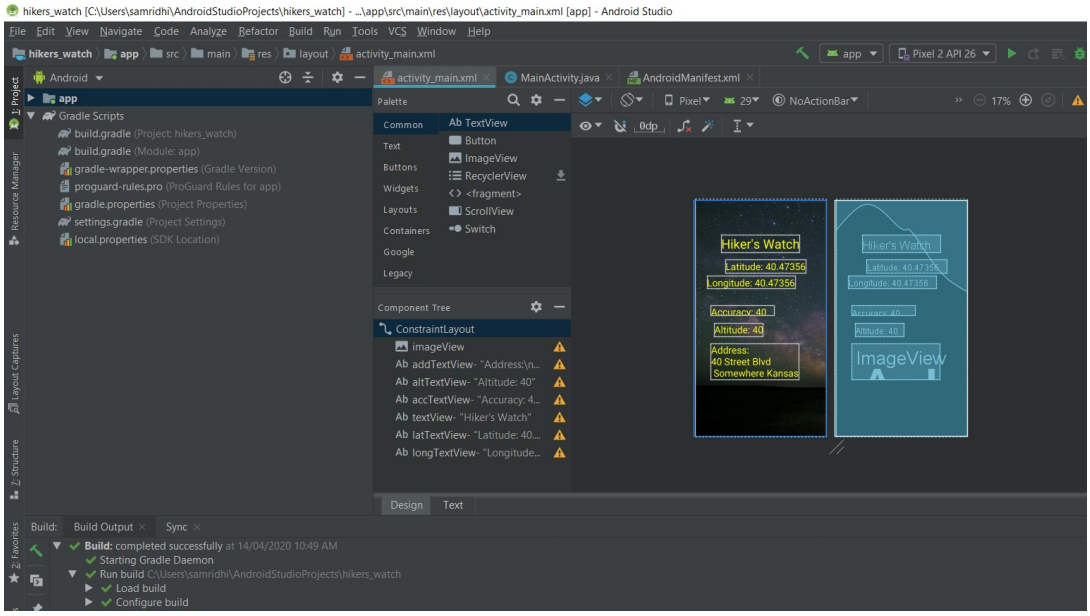

Fig 12 After the implementation of views and buttons

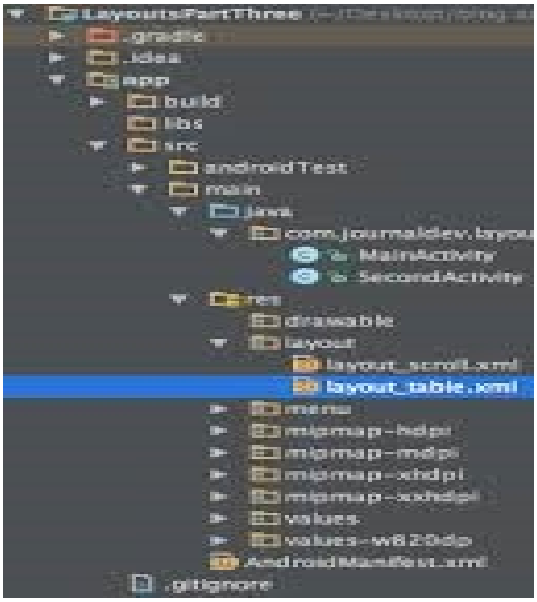

Fig 13 Text view buttons

4.2 SETTING UP THE EMULATOR

The next step would be to set up an android virtual device for your app. Without this, the code won't be synced on to your app. An emulator set up is required for this. I had to change my BIOS settings to be able to set up the emulator. The Android Emulator simulates Android devices on your computer so that you can test your application on a variety of devices and Android API levels without needing to have each physical device.

The emulator provides almost all of the capabilities of a real Android device. You can simulate incoming phone calls and text messages, specify the location of the device, simulate different network speeds, simulate rotation and other hardware sensors, access the Google Play Store, and much more.

Testing your app on the emulator is in some ways faster and easier than doing so on a physical device. For example, you can transfer data faster to the emulator than to a device connected over USB.

Fig 14 An image of the emulator

4.3 CODING THE APP

```
package com.samridhiknows.hikers_watch;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

import java.util.List;
import java.util.Locale;

public class MainActivity extends AppCompatActivity {

    LocationManager locationManager;
    LocationListener locationListener;

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

        locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
        locationListener = new LocationListener() {
            @Override
            public void onLocationChanged(Location location) {
                Log.i("Location", location.toString());
            }

            @Override
            public void onStatusChanged(String provider, int status, Bundle
extras) {

            }

            @Override
            public void onProviderEnabled(String provider) {

            }

            @Override
            public void onProviderDisabled(String provider) {

            }
        };

        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED)
{
            ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
        } else {
            locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
0, 0, locationListener);
            Location lastKnownLocation =
locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
            if (lastKnownLocation != null) {
```

```
                updateLocationInfo(lastKnownLocation);

            }
        }
    }

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults);

        if (grantResults.length>0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            startListening();
        }
    }

    public void startListening() {
        if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED)
{
            locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
0, 0, locationListener);


        }

    }

    public void updateLocationInfo(Location location) {
        TextView latTextView = findViewById(R.id.latTextView);
        TextView longTextView = findViewById(R.id.longTextView);
        TextView acctTextView = findViewById(R.id.accTextView);
        TextView altTextView = findViewById(R.id.altTextView);
        TextView addTextView = findViewById(R.id.addTextView);

        latTextView.setText("Latitude: " +
Double.toString(location.getLatitude()));
        longTextView.setText("Longitude: " +
Double.toString(location.getLongitude()));
        acctTextView.setText("Accuracy: " +
Double.toString(location.getAccuracy()));
        altTextView.setText("Altitude: " +
Double.toString(location.getAltitude()));

        String address = "Could not find address :(";
        Geocoder geocoder = new Geocoder(this, Locale.getDefault());

        try{
            List<Address>  listAddresses =
geocoder.getFromLocation(location.getLatitude(), location.getLongitude(), 1);

            if (listAddresses != null && listAddresses.size() > 0) {
                address = "Address: \n";

                if(listAddresses.get(0).getThoroughfare() != null) {
                    address += listAddresses.get(0).getThoroughfare() + "\n";
                }

                if(listAddresses.get(0).getLocality() != null) {
                    address += listAddresses.get(0).getLocality() + " ";
                }

                if(listAddresses.get(0).getPostalCode() != null) {
                    address += listAddresses.get(0).getPostalCode() + " ";
                }

                if(listAddresses.get(0).getAdminArea() != null) {
                    address += listAddresses.get(0).getAdminArea();
                }
```

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

        }

        addTextView.setText(address);

    }
}
```

CHAPTER 5

SAMPLE OUTPUT

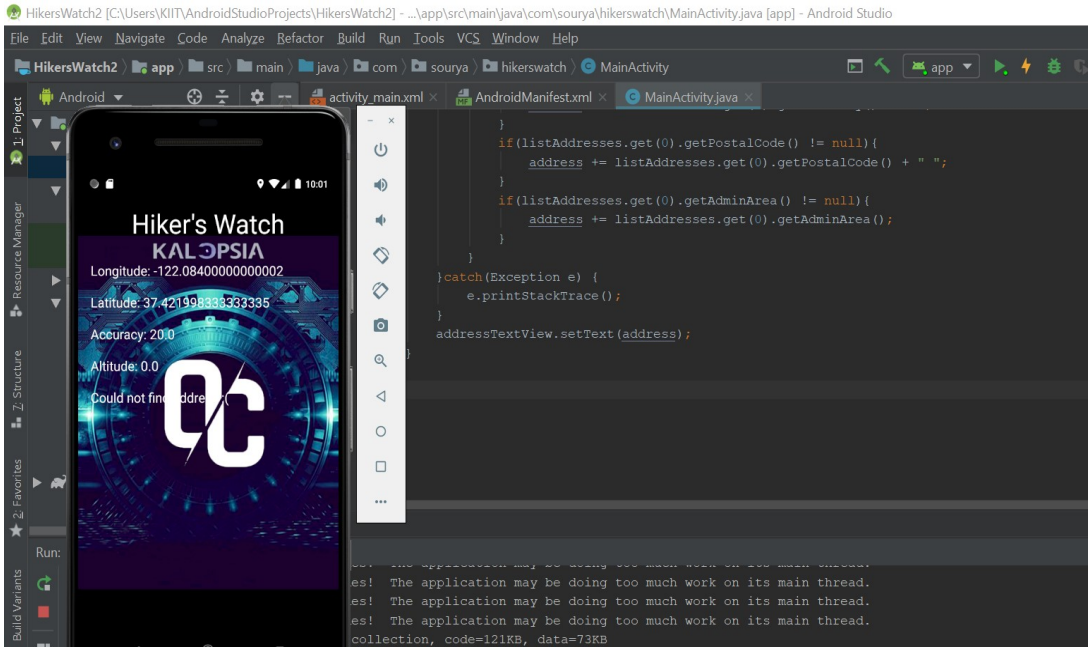

Fig 15 A sample output, when app could not find location

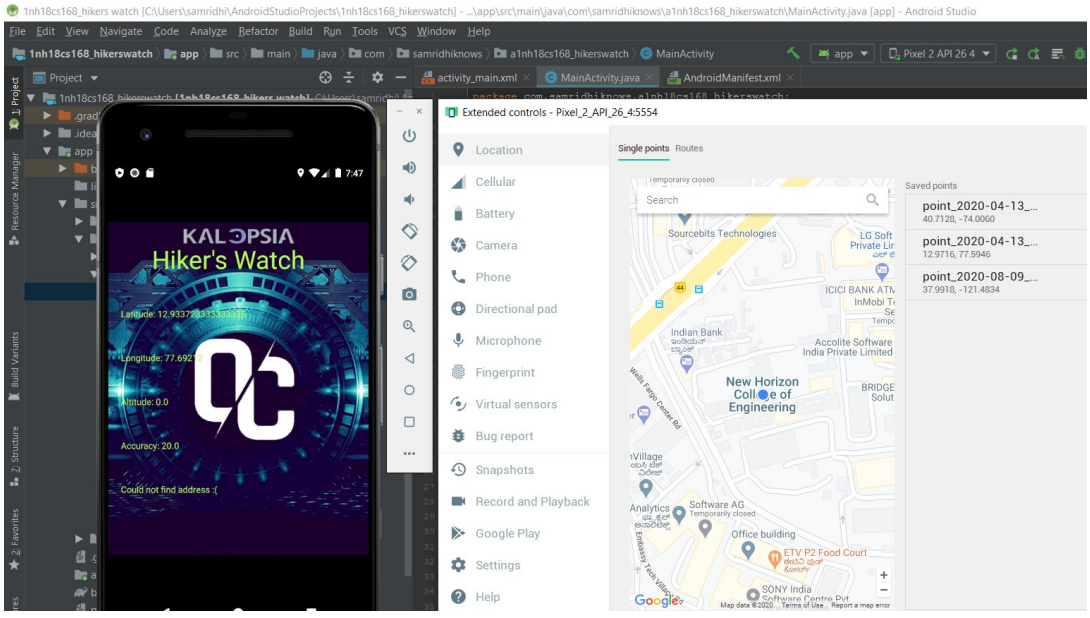

Fig 16, showing location of New Horizon College of Engineering

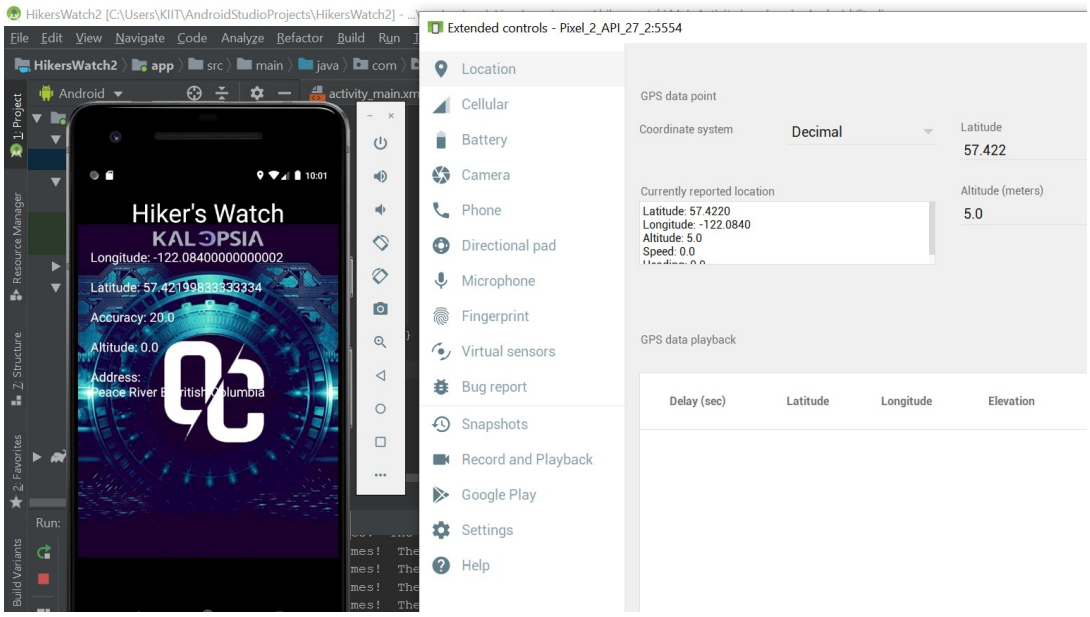

Fig 17 Sample output 2, showing a different location

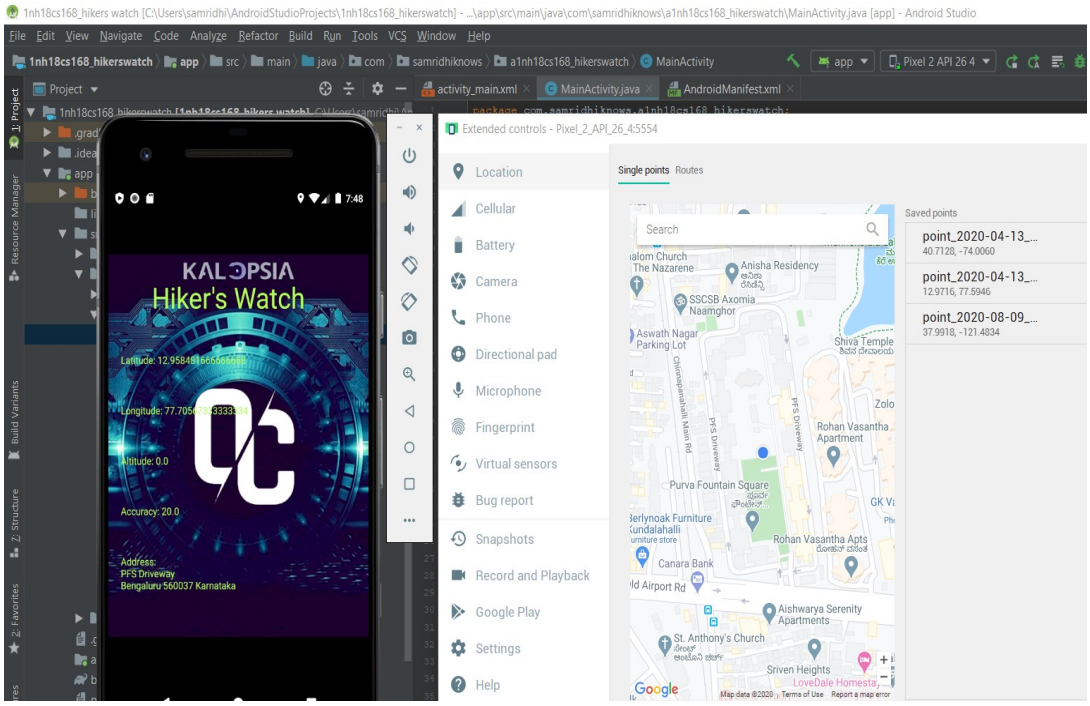

Fig 18 Sample output 4, app showing location of Purva Fountain Square, Bangalore

CHAPTER 6

CONCLUSION

Maps have changed our lives. Due to the several applications, we are able to travel anywhere around the world without being dependent much on any other individual. Google maps provides us with satellite imagery, and helps us in route planning. The sole purpose of this project was to demonstrate the importance of knowing location and the use of maps, by tuning in the Java programming language concepts.

By using the object oriented concepts of Java, we have successfully managed to design a fully fledged Android Studio app. The code explains the implementation of several Java concepts like the Inheritance, Polymorphism in the form of overriding, Exception handling, classes, objects, abstraction and many more.

Not only this, but this project has taught me to design an app. Apart from the coding, I have learnt to make use of the emulator and Google APIs.

CHAPTER 7

REFERENCES

1) Herbert Schildt, Java™: The Complete Reference, McGraw-Hill, Tenth Edition, 2018 .

2) https://www.udemy.com/course/the-complete-android-oreo-developer-course/learn/lecture/8339510#questions

3) https://developer.android.com/training/basics/firstapp

4) https://developer.android.com/reference/android/app/Activity

5) https://www.tutorialspoint.com/java/index.htm

ORIGINALITY REPORT

14%	%	14%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Reto Meier, Ian Lake. "Professional Android®", Wiley, 2018 Publication	2%
2	Mark Murphy. "Chapter 36 Basic Service Patterns", Springer Science and Business Media LLC, 2011 Publication	2%
3	Andres Calvo. "Beginning Android Wearables", Springer Science and Business Media LLC, 2015 Publication	2%
4	"Advances in Computational Intelligence, Security and Internet of Things", Springer Science and Business Media LLC, 2020 Publication	1%
5	Guliz Seray Tuncay, Soteris Demetriou, Carl A. Gunter. "Draco", Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16, 2016 Publication	1%

6 Raghav Sood. "Pro Android Augmented Reality", Springer Science and Business Media LLC, 2012 — 1%

Publication

7 John Hunt. "Essential JavaBeans fast", Springer Science and Business Media LLC, 1998 — 1%

Publication

8 Sung-Soo Kim, Chunglae Cho, Jongho Won. "A collaboration middleware for service scalability in peer-to-peer systems", 2015 International Conference on High Performance Computing & Simulation (HPCS), 2015 — 1%

Publication

9 "A WebGIS Application for Cloud Storm Monitoring", Communications in Computer and Information Science, 2016. — 1%

Publication

10 Ismail Hababeh, Sahel Alouneh, Ala F. Khalifeh. "A Position Aware Mobile Application for E- Health Services", 2016 7th International Conference on Intelligent Systems, Modelling and Simulation (ISMS), 2016 — 1%

Publication

11 Ted Hagos. "Android Studio IDE Quick Reference", Springer Science and Business Media LLC, 2019 — 1%

Publication

12 Abhijeet Banerjee, Abhik Roychoudhury. "Automated re-factoring of Android apps to enhance energy-efficiency", Proceedings of the International Workshop on Mobile Software Engineering and Systems - MOBILESoft '16, 2016 <1%
Publication

13 Onur Cinar. "Chapter 8 Sensors and Location", Springer Science and Business Media LLC, 2015 <1%
Publication

14 Dietmar P. F. Möller, Roland E. Haas. "Chapter 7 Mobile Apps for the Connected Car", Springer Science and Business Media LLC, 2019 <1%
Publication

15 Ying Pu, , Wai-Choong Wong, and Huaqun Guo. "Implementation and evaluation of remote tracking system", 5th International Conference on Computer Sciences and Convergence Information Technology, 2010. <1%
Publication

16 Satya Komatineni, Dave MacLean, Sayed Y. Hashimi. "Pro Android 3", Springer Science and Business Media LLC, 2011

17 Mario Zechner, J. F. DiMarzio, Robert Green. "Beginning Android Games", Springer Science <1%

and Business Media LLC, 2016

Publication

---

18 Dave Smith, Erik Hellman. "Android Recipes", Springer Science and Business Media LLC, 2016 <1%

Publication

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	On		

HIKER'S WATCH, AN APP FOR THE ADVENTUROUS

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29