**Mini Project**

4

**1.2.5.**

**SQLite**

Extremely small (< 500kb) relational database management system, isintegrated in Android. It is based on function calls and single file, where alldefinitions, tables and data are stored. This simple design is more than suitable for aplatform such as Android.

**1.2.6.**

**Handset Layouts**

The platform is adaptable to both larger, VGA, 2D graphics library, 3Dgraphics library based on OpenGL ES 1.0 specifications, traditional smart phonelayouts. An underlying 2D graphics engine is also included. Surface Managermanages access to the display subsystem and seamlessly composites 2D and 3Dgraphic layers from multiple applications

**1.2.7.**

**Data Storage**

SQLite is used for structured data storage .SQLite is a powerful andlightweight relational database engine available to all applications.

**1.2.8.**

**Connectivity**

Android supports a wide variety of connectivity technologies including GSM,CDMA, Bluetooth, EDGE, EVDO, 3G and Wi-Fi.

**1.2.9.**

**Messaging**

SMS, MMS, and XMPP are available forms of messaging including threadedtext messaging.

**1.2.10.**

**Web Browser**

The web browser available in Android is based on the open-source WebKitapplication framework. It includes LibWebCore which is a modern web browserengine which powers both the Android browser and an embeddable web view.

**1.2.11.**

**Java Virtual Machine**

Software written in Java can be compiled into Dalvik byte codes and executedin the Dalvik virtual machine, which is a specialized VM implementation designed formobile device use, although not technically a standard Java Virtual Machine.

5

**1.2.12.**

**Media Support**

Android will support advanced audio/video/still media formats such asMPEG-4, H.264, MP3, and AAC, AMR, JPEG, PNG, GIF.

**1.2.13.**

**Additional Hardware Support**

Android is fully capable of utilizing video/still cameras, touch screens, GPS,compasses, accelerometers, and accelerated 3D graphics.

**1.2.14.**

**Development Environment**

Includes a device emulator, tools for debugging, memory and performanceprofiling, and a plug-in for the Eclipse IDE. There are a number of hardwaredependent features, for instance, a huge media and connections support, GPS,improved support for Camera and simply GSM telephony. A great work was donefor the developers to start work with Android using device emulator, tools fordebugging and plug-in for Eclipse IDE

**On**

**Cafe Mitra: The Smart Cafe Assistant!**

|  |  |
| --- | --- |
| **NAME** | **ROLL NO** |
| **ANIRUDDHA CHAKRAVARTY** | **2173042** |
| **KALPAK JIBHAKATE** | **2173098** |
| **AVINASHDEV GARUDAPALLI** | **2173058** |

**Project Guide**

Prof. Nagesh N. Jadhav



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**MIT-ADT UNIVERSITY, SCHOOL OF ENGINEERING,**

**RAJBAUG, LONI, INDIA**

**MIT School of Engineering**

**Department of Computer Science & Engineering**

**RAJBAUG, LONI, INDIA**



***CERTIFICATE***

This is to certify that the Mini Project entitled **“Cafe Mitra: The Smart Cafe Assistant“,** submitted by **Aniruddha Chakravarty(2173042), Avinashdev Garudapalli (2173058)**, **and Kalpak Jibhakate(2173098)** is a record of bonafide work carried out by them in the partial fulfillment of the requirement for the award of Degree of B.Tech (CSE) at MIT School of Engineering, Rajbaug, Loni, Pune under the MIT ADT University. This work is done during the year 2018-19, Semester IV.

Date: 24 / 04 / 2019

----------------------------------------- ---------------------------------------------

Prof. Nagesh Jadhav Dr. Rajneeshkaur Sachdeo Bedi

Project Guide Dean Engineering,

MIT School of Engineering Head , Department of CSE

Loni Kalbhor, Pune MIT School of Engineering

**ACKNOLEDGEMENT**

We take great pleasure in submitting this project report based on **“****Cafe Mitra: The Smart Cafe Assistant“.** I wish to express true sense of gratitude towards our guide, Prof. Nagesh Jadhav who at very discrete stages in study of this project, contributed his valuable guidance and help to solve every problem that arose.

We also thank our H.O.D. Dr. Rajneeshkaur Sachdeo-Bedi, for providing the necessary facilities for the completion of this project work in our college.

Most importantly we would like to express my sincere gratitude towards our respective families for always being there when needed them the most. We would also like to thank all our friends for the undying support shown by them during the preparation of this project. We feel as if we owe all of our success to them.

**Table of Contents**

# Contents

|  |  |
| --- | --- |
| **Sr. No.** | **Contents** |
| 1 | Abstract |
| 3 | **List of Figures** |
| 4 | 1. INTRODUCTION 2. PROBLEM DEFINITION 3. FEATURES OF PROJECT 4. PLATFORM / TECHNOLOGY 5. FLOWCHART 6. OUTPUT 7. CODE 8. GLOSSARY 9. REFERENCES |
| 5 | 1. CONCLUSION AND FUTURE ENHANCEMENTS |
| 6 | **Glossary** |
| 7 | Reference & Bibliography |
| 8 | **Annexure A** |

**ABSTRACT**

**“Cafe Mitra: The Smart Cafe Assistant“** Is an Android based application for implementing Android based technologies into a domain where traditionally, very less technology has been used.

Android is a software stack for mobile devices that includes an operating system, middle ware and key applications.

Android is a software platform and operating system for mobile devices based on the Linux operating system and developed by Google and the Open Handset Alliance. It allows developers to write managed code in a Java-like language that utilizes Google-developed Java libraries,but does not support programs developed in native code. The unveiling of the Android platform on 5 November 2007 was announced with the founding of the Open Handset Alliance, a consortium of 34 hardware,software and telecom companies devoted to advancing open standards for mobile devices. When released in 2008, most of the Android platform will be made available under the Apache free-software and open-source license

The project aims to incorporate state-of-the-art techniques in Android along with the goal of achieving high accuracy with a real-time performance. This application will aim to save a user’s order requests automatically, offer suggestions when the user is stuck and come full of links for how coffee is made, and where the beans are sourced from, amongst other things and will offer other resources to help the user’s journey of selecting his/her coffee a lot less painful !

In the application, we used a set of options that are known to be common to all cafes. Cafes as such offer a a very similar, standardized menu to their customers, and we have offered standardized alternatives. The customer is rightfully the king, and he gets to choose whatever he wants on the standardized menu.

Being super intimidated by this line-up of problems that nearly every restaurant faces and having been putting off tackling them for quite sometimes, we finally began to dig in. Tracking down tutorials, reading every article we could find and running circles of confusion around ourselves day in day out.

The answer was to build a simple app that would help us achieve our goals and keep us well-practiced in our core skills. So we decided to bring the two ideas together.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Fig No.** | **Figure Name** | **Page No.** |
| 1. | ABSTRACT |  |
| 2. | LIST OF TABLES |  |
| 3. | LIST OF FIGURES |  |
| 4. | 1. INRODUCTION  2. PROBLEM DEFINATION  3. FEATURES OF PROJECT  4. PLATFORM / TECHNOLOGY  5. FLOWCHART  6. OUTPUT  7.CODE  8.REFERENCES |  |
| 5. | CONCLUSION & FUTURE ENHANCEMENTS |  |
| 6 | GLOSSARY |  |
| 7 | REFERENCE & BIBLIOGRPHY |  |
| 8 | ANNEXURE A:Weekly Reporting Sheet |  |

**INTRODUCTION**

**“Cafe Mitra: The Smart Cafe Assistant“** Is an Android based application for implementing Android based technologies into a domain where traditionally, very less technology has been used. It aims to reduce the distance between the customer at the restaurant and the servers.

The project aims to incorporate state-of-the-art techniques in Android along with the goal of achieving high accuracy with a real-time performance. This application will aim to save a user’s order requests automatically, offer suggestions when the user is stuck and come full of links for how coffee is made, and where the beans are sourced from, amongst other things and will offer other resources to help the user’s journey of selecting his/her coffee a lot less painful !

The answer was to build a simple app that would help us achieve my goals and keep me well-practiced in our core skills. So we decided to bring the two ideas together. To me, the best way to learn something, is to relate it to something you love. So we were building this app, and having a blast doing it, and were also developing content for it.

Let us acknowledge outright that this app is nothing groundbreaking. “**Cafe Mitra”** intends to be extremely minimalistic, and hones in on a very specific set of topics. Plus, this is not meant to be the next big thing. Building this app was just our way of making learning this stuff fun. If it ends up being a valuable resource for even one other person along the way, all the better.

**PROBLEM DEFINITION**

“**Cafe Mitra”** , is an application which integrates technology with a problem that is often found in restaurants. It aims to eliminate the requirement of a waiter, and directly sends an invoice to the customer. As a result, simpler errors such as tallying mistakes are reduced, and the system becomes faster, more responsive and efficient overall.

The complete system will be run as an Android based application , or as a sandbox on a browser.

This application will aim to save a user’s order requests automatically, offer suggestions when the user is stuck and come full of links for how coffee is made, and where the beans are sourced from, amongst other things and will offer other resources to help the user’s journey of selecting his/her coffee a lot less painful !

**H/W AND S/W Resources:**

The end application will be run on Android. The hardware pre-requisites will include 3GB RAM, and 32GB storage space .The software requirements will be a 64bit Android based operating system (versions 6.0.0 and above)

Alternatively, **“Cafe Mitra”** could also be run on the browser as a Java Scripts and browser sandbox.

**FEATURES OF PROJECT**

**Application Framework**

Unlike other embedded mobile environments, Android applications are all equal, for instance, an applications which come with the phone are no different than those that any developer writes. The framework is supported by numerous open source libraries such as openssl, SQLite and libc. It is also supported by the Android core libraries. From the point of security, the framework is based on UNIX file system permissions that assure applications have only those abilities that mobile phone owner gave them at install time.

**Dalvik Virtual Machine**

It is extremely low-memory based virtual machine, which was designed especially for Android to run on embedded systems and work well in low power situations. It is also tuned to the CPU attributes. The Dalvik VM creates a special file format (.DEX) that is created through build time post processing. Conversion between Java classes and .DEX format

is done by included “dx” tool.

**Integrated Browser**

Google made a right choice on choosing WebKit as open source web browser. They added a two pass layout and frame flattening. Two pass layout loads a page without waiting for blocking elements, such as external CSS or external JavaScript and after a while renders again with all resources downloaded to the device. Frame flattening converts founded frames into single one and loads into the browser. These features increase speed and usability browsing the internet via mobile phone.

**Optimized Graphics**

As Android has 2D graphics library and 3D graphics based on OpenGLES 1.0, possibly we will see great applications like Google Earth and spectacular games like Second Life, which come on Linux version. At this moment,the shooting legendary 3D game Doom was presented using Android on the mobile phone.

**SQLite**

Extremely small (< 500kb) relational database management system, is integrated in Android. It is based on function calls and single file, where all definitions, tables and data are stored. This simple design is more than suitable for a platform such as Android.

.

**Handset Layouts**

The platform is adaptable to both larger, VGA, 2D graphics library, 3Dgraphics library based on OpenGL ES 1.0 specifications, traditional smart phonelayouts. An underlying 2D graphics engine is also included. Surface Managermanages access to the display subsystem and seamlessly composites 2D and 3Dgraphic layers from multiple applications

**Development Environment**

Includes a device emulator, tools for debugging, memory and performanceprofiling, and a plug-in for the Eclipse IDE. There are a number of hardwaredependent features, for instance, a huge media and connections support, GPS,improved support for Camera and simply GSM telephony. A great work was donefor the developers to start work with Android using device emulator, tools fordebugging and plug-in for Eclipse IDE.

**Additional Hardware Support**

Android is fully capable of utilizing video/still cameras, touch screens, GPS,compasses, accelerometers, and accelerated 3D graphics

**Media Support**

Android will support advanced audio/video/still media formats such asMPEG-4, H.264, MP3, and AAC, AMR, JPEG, PNG, GIF

**Java Virtual Machine**

Software written in Java can be compiled into Dalvik byte codes and executedin the Dalvik virtual machine, which is a specialized VM implementation designed formobile device use, although not technically a standard Java Virtual Machine.

**Web Browser**

The web browser available in Android is based on the open-source WebKitapplication framework. It includes LibWebCore which is a modern web browserengine which powers both the Android browser and an embeddable web view

**Messaging**

SMS, MMS, and XMPP are available forms of messaging including threadedtext messaging

**Connectivity**

Android supports a wide variety of connectivity technologies including GSM,CDMA, Bluetooth, EDGE, EVDO, 3G and Wi-Fi

**PLATFORM/TECHONOLGY USED**

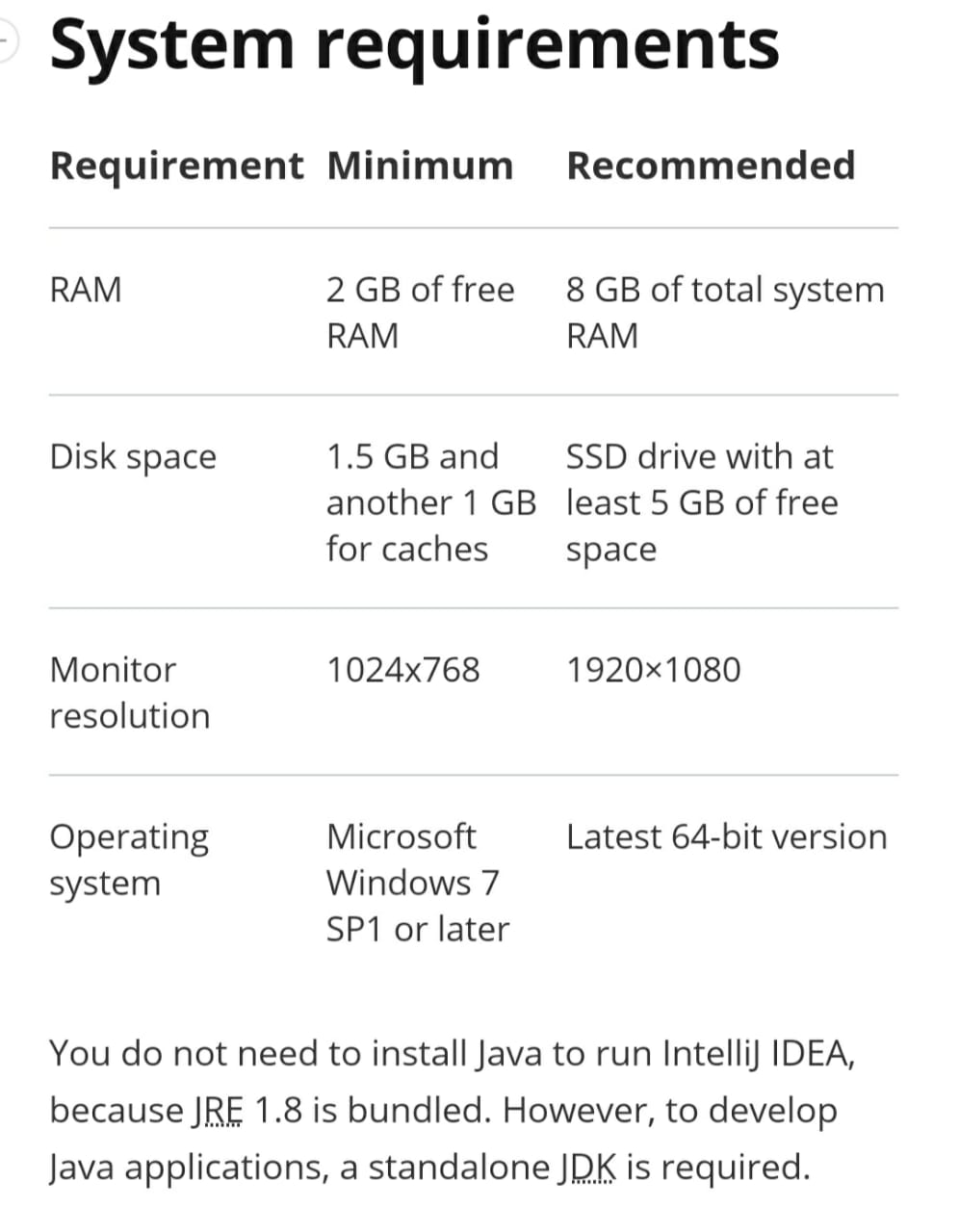
**1.** A PC/Laptop with Windows 7/8/10 and Linux

**2.** Java SE 1.8/10

**3.** Download JDK/JRE with given following link <http://www.oracle.com/technetwork/java/javase/downloads/index.html> (WINDOWS)

<https://websiteforstudents.com/install-oracle-java-jre-jdk-on-ubuntu-18-04-lts-beta/> (LINUX)

**4.** IntelliJ IDEA 2019.1.1 or any other equivalent IDE software such as netbeans



[https://www.jetbrains.com/idea/download/#section=windows](https://www.jetbrains.com/idea/download/" \l "section=windows)

The end application was planned to be run on Android. The hardware pre-requisites would have included 3 GB RAM, and 32 GB storage space. The software requirements will be a 64 bit Android based operating system (versions 6.0.0 and above)

5]  **Install SDK Platforms; Install Android Studio:** 1. Install JDK 6 or later

First, install Oracle JDK 8

sudo add-apt-repository ppa:webupd8team/java

sudo apt-get update

sudo apt-get install oracle-java8-installer

sudo apt-get install oracle-java8-set-default

To make sure, it’s installed successfully, open a terminal and type

javac -version

Set the JAVA\_HOME environment variable to the location of your JDK installation.

(http://www.wikihow.com/Set-Up-Your-Java\_Home-Path-in-Ubuntu)

2. **Download and install Android Studio**

Download the Android Studio package for Linux and extract it somewhere (e.g home directory).

To launch Android Studio, open a terminal, navigate to the android-studio/bin/ directory, and execute studio.sh.

cd android-studio/bin

./studio.sh

Select whether you want to import previous Android Studio settings or not, then click OK.

Set the ANDROID\_HOME environment variable to the location of your Android SDK installation

sudo gedit ~/.bashrc

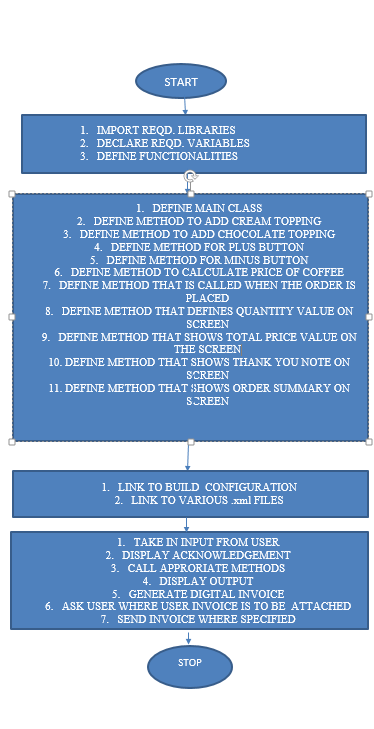
export ANDROID\_HOME=/home/user\_directory/Android/Sdk

export PATH=${PATH}:$ANDROID\_HOME/tools:$ANDROID\_HOME/platform-tools

export JAVA\_HOME=/usr/lib/jvm/java-8-oracle

3. **Install SDK Platforms**

You need to install some SDK before you jump into building android apps. Click on Configure -> SDK Manager to open Android SDK Manager. Select the latest API (to test against target build, e.g API 19 (Android 4.4.2)) and some packages in Extras (Android Support Library and Android Support Repository). Then install the selected packages.

**FLOWCHART**

**CODE**

**MAIN ACTIVITY:**

package com.example.android.coffeetime;

import android.content.Intent;

import android.graphics.Color;

import android.net.Uri;

import android.os.Bundle;

import android.support.v7.app.AppCompatActivity;

import android.view.View;

import android.widget.CheckBox;

import android.widget.EditText;

import android.widget.TextView;

import android.widget.Toast;

import java.text.NumberFormat;

public class MainActivity extends AppCompatActivity {

int quantity = 0;

boolean hascream;

boolean haschocolatecream;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

/\*\*

\* **This method is called when cream topping is added**

\*/

public void creamtopping() {

CheckBox creamcheck = (CheckBox) findViewById(R.id.creamcheck);

hascream = creamcheck.isChecked();

}

/\*\*

\* **This method is called when chocolate topping is added**

\*/

public void chococlatetopping() {

CheckBox chocolatecheck = (CheckBox) findViewById(R.id.chocolatcheck);

haschocolatecream = chocolatecheck.isChecked();

}

/\*\*

\* **This method is called when the plus button is pressed.**

**\*/**

public void increment(View view)

**This method is called when the minus button is pressed**) {

if (quantity == 100) {

Toast.makeText(getApplicationContext(), "This is the maximum amount of Coffee u can order!! ", Toast.LENGTH\_SHORT).show();

return;

}

quantity = quantity + 1;

TextView welcome = (TextView) findViewById(R.id.welcome);

welcome.setText(getResources().getText(R.string.welcome));

welcome.setTextColor(getResources().getColor(R.color.welcome));

welcome.setTextSize(20);

TextView ordersummarytext = (TextView) findViewById(R.id.price);

ordersummarytext.setText(getResources().getText(R.string.price));

display(quantity);

total(calculatePrice());

}

/\*\*

\* **This method is called when the minus button is pressed.**

\*/

public void decrement(View view) {

if (quantity == 1) {

Toast.makeText(getApplicationContext(), "This is the minimum amount of Coffee u can order!! ", Toast.LENGTH\_SHORT).show();

return;

}

quantity = quantity - 1;

TextView welcome = (TextView) findViewById(R.id.welcome);

welcome.setText(getResources().getText(R.string.welcome));

welcome.setTextColor(getResources().getColor(R.color.welcome));

welcome.setTextSize(20);

TextView ordersummarytext = (TextView) findViewById(R.id.price);

ordersummarytext.setText(getResources().getText(R.string.price));

display(quantity);

total(calculatePrice());

}

/\*\*

\* **This method calculate price of coffee**

\*/

private int calculatePrice() {

int pricecream = 10;

int pricechocolate = 25;

int pricePerCup = 50;

creamtopping();

chococlatetopping();

if (hascream) {

pricePerCup = pricePerCup + pricecream;

} // add the price of whipped cream

if (haschocolatecream) {

pricePerCup = pricePerCup + pricechocolate;

} // add the price of chocolate topping

return quantity \* pricePerCup;

}

/\*\*

\* **This method is called when the order button is pressed**.

\*/

public void order(View view) {

ordersummary();

bye();

}

/\*\*

\* **This method displays the given quantity value on the screen.**

\*/

private void display(int number) {

TextView quantity = (TextView) findViewById(R.id.quantity);

quantity.setText("" + number);

}

/\*\*

\* **This method is for the total price value on the screen.**

\*/

private void total(int number) {

TextView total = (TextView) findViewById(R.id.total);

total.setText(NumberFormat.getCurrencyInstance().format(number));

total.setTextColor(getResources().getColor(R.color.total));

}

/\*\*

\* **This method displays the Thank you note on the screen.**

\*/

private void bye() {

TextView bill = (TextView) findViewById(R.id.welcome);

bill.setText(getResources().getText(R.string.bye));

bill.setTextSize(25);

bill.setTextColor(Color.YELLOW);

}

/\*\*

\* **This method displays the order summary on the screen.**

\*/

private void ordersummary() {

EditText editname = (EditText) findViewById(R.id.name);

String name = editname.getText().toString();

String orderSummary = ("Name : " + name + "\nAdd Whipped Cream : " + hascream + "\nAdd Chocolate : " + haschocolatecream + "\nQuantity : " + quantity + "\nTotal : " + "Rs. " + calculatePrice() + "\nThank YOU!!!");

Intent intent = new Intent((Intent.ACTION\_SENDTO));

intent.setData(Uri.parse("mailto:"));

intent.putExtra(Intent.EXTRA\_SUBJECT, "COFFEE ORDER for " + name);

intent.putExtra(Intent.EXTRA\_TEXT, orderSummary);

if (intent.resolveActivity(getPackageManager()) != null) {

startActivity(intent);

}

}

}

**BUILD CONFIGURATION:**

package com.example.android.coffeetime;

public final class BuildConfig {

public static final boolean DEBUG = Boolean.parseBoolean("true");

public static final String APPLICATION\_ID = "com.example.android.coffeetime";

public static final String BUILD\_TYPE = "debug";

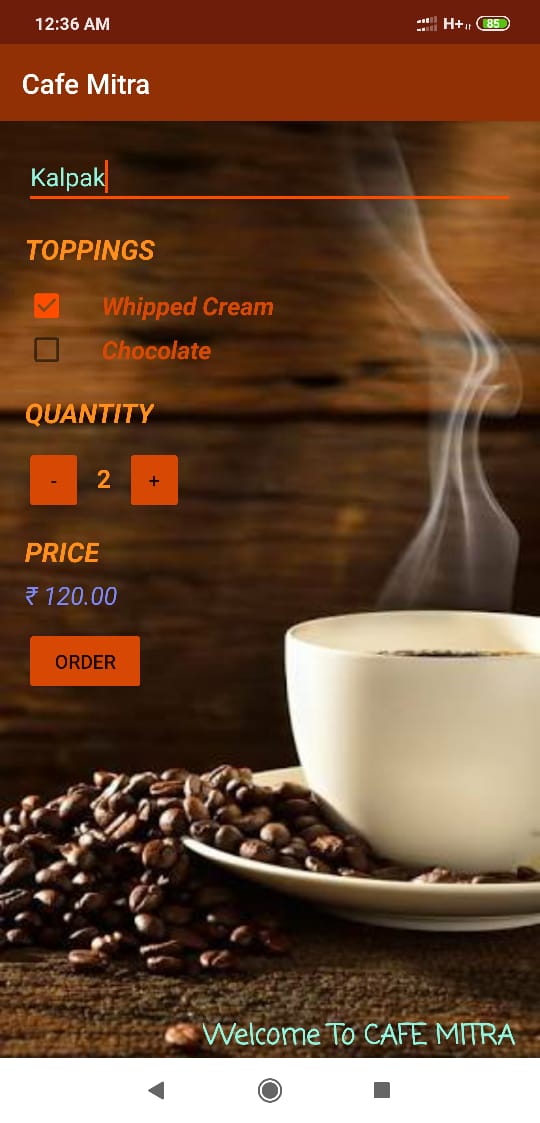
public static final String FLAVOR = "";

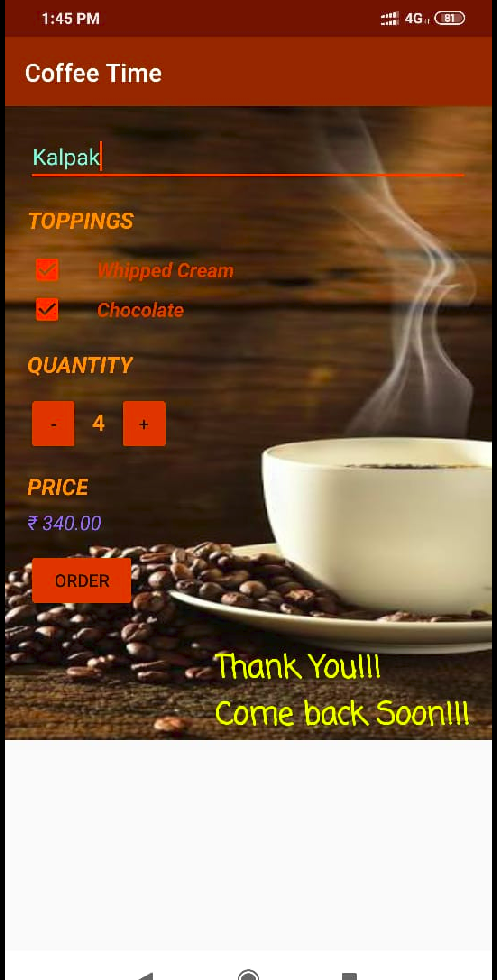
public static final int VERSION\_CODE = 1;

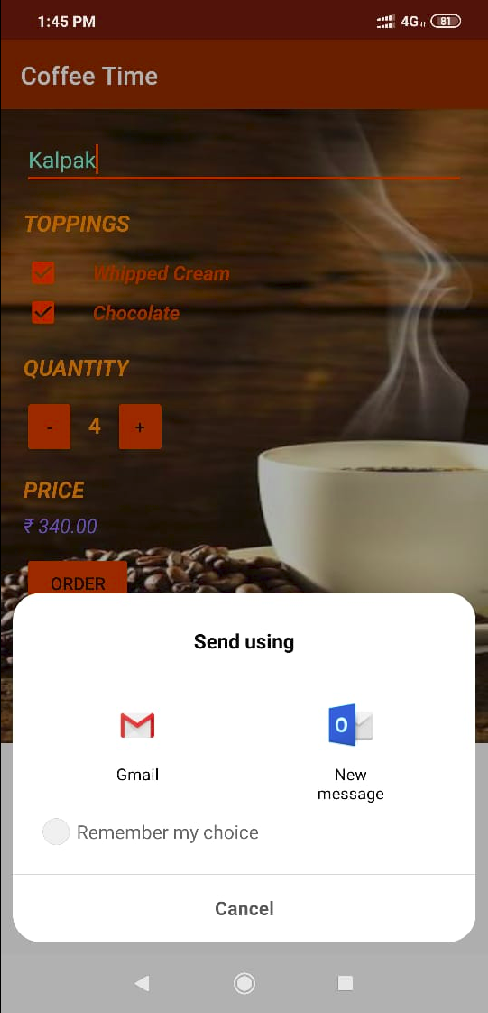
public static final String VERSION\_NAME = "1.0";

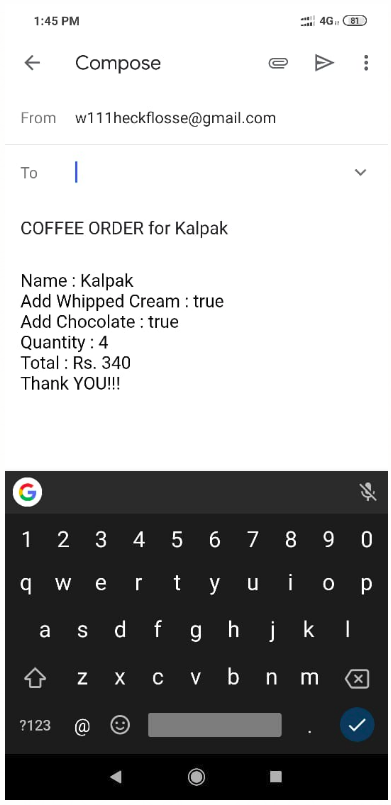
}

**OUTPUT**



****

****

****

**CONCLUSION**

We have built “**Cafe Mitra”** , which is an application which integrates a technology with a problem that is often found in restaurants. It aims to eliminate the requirement of a waiter, and directly sends an invoice to the customer. As a result, simpler errors such as tallying mistakes are reduced, and the system becomes faster, more responsive and efficient overall.

The complete system will be run as an Android based application , or as a sandbox on a browser.

This project is still under developing stage as we are trying out different algorithms to improve its functioning and make it more efficient compared to what it already is.

**GLOSSARY (For Java Terminologies)**

**abstract class**

A class that contains one or more abstract methods , and therefore can never be instantiated. Abstract classes are defined so that other classes can extend them and make them concrete by implementing the abstract methods.

**abstract method**

A method that has no implementation.

**Abstract Window Toolkit (AWT)**

A collection of graphical user interface (GUI) components that were implemented using native-platform versions of the components. These components provide that subset of functionality which is common to all native platforms. Largely supplanted by the Project Swing component set. See also Swing .

**access control**

The methods by which interactions with resources are limited to collections of users or programs for the purpose of enforcing integrity, confidentiality, or availability constraints.

**ACID**

The acronym for the four properties guaranteed by transactions: atomicity, consistency, isolation, and durability.

**actual parameter list**

The arguments specified in a particular method call. See also formal parameter list .

**API**

Application Programming Interface. The specification of how a programmer writing an application accesses the behavior and state of classes and objects.

**applet**

A component that typically executes in a Web browser, but can execute in a variety of other applications or devices that support the applet programming model.

**argument**

A data item specified in a method call. An argument can be a literal value, a variable, or an expression.

**array**

A collection of data items, all of the same type, in which each item's position is uniquely designated by an integer. binary operator An operator that has two arguments.bitThe smallest unit of information in a computer, with a value of either 0 or 1.

**bitwise operator**

An operator that manipulates the bits of one or more of its operands individually and in parallel. Examples include the binary logical operators (&, |, ^), the binary shift operators (<<, >>, >>>) and the unary one's complement operator (~).

**block**

In the Java programming language, any code between matching braces. Example: { x = 1; }.

**boolean**

Refers to an expression or variable that can have only a true or false value. The Java programming language provides the boolean type and the literal values true and false.

**break**

A Java keyword used to resume program execution at the statement immediately following the current statement. If followed by a label, the program resumes execution at the labeled statement.

**byte**

A sequence of eight bits. Java provides a corresponding byte type.

**For C:**

**case**

A Java keyword that defines a group of statements to begin executing if a value specified matches the value defined by a preceding switch keyword.

**casting**

Explicit conversion from one data type to another.

**catch**

A Java keyword used to declare a block of statements to be executed in the event that a Java exception, or run time error, occurs in a preceding try block.

**char**

A Java keyword used to declare a variable of type character.

**class**

In the Java programming language, a type that defines the implementation of a particular kind of object. A class definition defines instance and class variables and methods, as well as specifying the interfaces the class implements and the immediate superclass of the class. If the superclass is not explicitly specified, the superclass will implicitly be Object.

**class method**

A method that is invoked without reference to a particular object. Class methods affect the class as a whole, not a particular instance of the class. Also called a static method . See also instance method .

**class variable**

A data item associated with a particular class as a whole--not with particular instances of the class. Class variables are defined in class definitions. Also called a static field . See also instance variable .

**classpath**

An environmental variable which tells the Java virtual machine 1 and Java technology-based applications where to find the class libraries, including user-defined class libraries.

In a program, explanatory text that is ignored by the compiler. In programs written in the Java programming language, comments are delimited using // or /\*... \*/.

**commit**

The point in a transaction when all updates to any resources involved in the transaction are made permanent.

**compilation unit**

The smallest unit of source code that can be compiled. In the current implementation of the Java platform, the compilation unit is a file.

**compiler**

A program to translate source code into code to be executed by a computer. The Java compiler translates source code written in the Java programming language into bytecode for the Java virtual machine 1

**constructor**

A pseudo-method that creates an object. In the Java programming language, constructors are instance methods with the same name as their class. Constructors are invoked using the new keyword.

**const**

A reserved Java keyword not used by current versions of the Java programming language.

**continue**

A Java keyword used to resume program execution at the end of the current loop. If followed by a label, continue resumes execution where the label occurs.

**conversational state**

The field values of a session bean plus the transitive closure of the objects reachable from the bean's fields. The transitive closure of a bean is defined in terms of the serialization protocol for the Java programming language, that is, the fields that would be stored by serializing the bean instance.

**CORBA**

Common Object Request Broker Architecture. A language independent, distributed object model specified by the Object Management Group (OMG).

**core class**

A public class (or interface) that is a standard member of the Java Platform. The intent is that the core classes for the Java platform, at minimum, are available on all operating systems where the Java platform runs. A program written entirely in the Java programming language relies only on core classes, meaning it can run anywhere. .

**core packages**

The required set of APIs in a Java platform edition which must be supported in any and all compatible implementations.

**For D:**

**declaration**

A statement that establishes an identifier and associates attributes with it, without necessarily reserving its storage (for data) or providing the implementation (for methods). See also definition.

**default**

A Java keyword optionally used after all case conditions in a switch statement. If all case conditions are not matched by the value of the switch variable, the default keyword will be executed.

**definition**

A declaration that reserves storage (for data) or provides implementation (for methods). See also declaration.

**delegation**

An act whereby one principal authorizes another principal to use its identity or privileges with some restrictions.

**deprecation**

Refers to a class, interface, constructor, method or field that is no longer recommended, and may cease to exist in a future version.

**derived from**

Class X is "derived from" class Y if class X extends class Y. See also subclass , superclass .

**distributed**

Running in more than one address space.

**distributed application**

An application made up of distinct components running in separate runtime environments, usually on different platforms connected through a network. Typical distributed applications are two-tier (client/server), three-tier (client/middleware/server), and n-tier (client/multiple middleware/multiple servers).

**do**

A Java keyword used to declare a loop that will iterate a block of statements. The loop's exit condition can be specified with the while keyword.

double

A Java keyword used to define a variable of type double.

**For E:**

**else**

A Java keyword used to execute a block of statements in the case that the test condition with the if keyword evaluates to false.

**Embedded Java Technology**

The availability of Java 2 Platform, Micro Edition technology under a restrictive license agreement that allows a licensee to leverage certain Java technologies to create and deploy a closed-box application that exposes no APIs.

**encapsulation**

The localization of knowledge within a module. Because objects encapsulate data and implementation, the user of an object can view the object as a black box that provides services. Instance variables and methods can be added, deleted, or changed, but as long as the services provided by the object remain the same, code that uses the object can continue to use it without being rewritten. See also instance variable , instance method.

**enum**

A Java keyword used to declare an enumerated type.

**enumerated type**

A type whose legal values consist of a fixed set of constants.

**exception**

An event during program execution that prevents the program from continuing normally; generally, an error. The Java programming language supports exceptions with the try, catch, and throw keywords. See also exception handler.

e**xception handler**

A block of code that reacts to a specific type of exception. If the exception is for an error that the program can recover from, the program can resume executing after the exception handler has executed.

**executable content**

An application that runs from within an HTML file. See also applet .

**Extends**

Class X extends class Y to add functionality, either by adding fields or methods to class Y, or by overriding methods of class Y. An interface extends another interface by adding methods. Class X is said to be a subclass of class Y. See also derived from .

For F

field

A data member of a class. Unless specified otherwise, a field is not static.

**final**

A Java keyword. You define an entity once and cannot change it or derive from it later. More specifically: a final class cannot be subclassed, a final method cannot be overridden and a final variable cannot change from its initialized value.

**finally**

A Java keyword that executes a block of statements regardless of whether a Java Exception, or run time error, occurred in a block defined previously by the try keyword.

**float**

A Java keyword used to define a floating point number variable.

**for**

A Java keyword used to declare a loop that reiterates statements. The programmer can specify the statements to be executed, exit conditions, and initialization variables for the loop.

**For G**:

generic

A class, interface, or method that declares one or more type variables. These type variables are known as type parameters. A generic declaration defines a set of parameterized types, one for each possible invocation of the type parameter section. At runtime, all of these parameterized types share the same class, interface, or method.

**goto**

This is a reserved Java keyword. However, it is not used by current versions of the Java programming language.

**group**

A collection of principals within a given security policy domain.

**GUI**

Graphical User Interface. Refers to the techniques involved in using graphics, along with a keyboard and a mouse, to provide an easy-to-use interface to some program.

**For H:**

**hexadecimal**

The numbering system that uses 16 as its base. The marks 0-9 and a-f (or equivalently A-F) represent the digits 0 through 15. In programs written in the Java pro

**For I:**

**IDL**

Interface Definition Language. APIs written in the Java programming language that provide standards-based interoperability and connectivity with CORBA (Common Object Request Broker Architecture).

identifier

The name of an item in a program written in the Java programming language.

If A Java keyword used to conduct a conditional test and execute a block of statements if the test evaluates to true.

**impersonation**

An act whereby one entity assumes the identity and privileges of another entity without restrictions and without any indication visible to the recipients of the impersonator's calls that delegation has taken place. Impersonation is a case of simple delegation.

**implements**

A Java keyword included in the class declaration to specify any interfaces that are implemented by the current class.

**import**

A Java keyword used at the beginning of a source file that can specify classes or entire packages to be referred to later without including their package names in the reference.

**inheritance**

The concept of classes automatically containing the variables and methods defined in their supertypes . See also superclass , subclass .

**instance**

An object of a particular class. In programs written in the Java programming language, an instance of a class is created using the new operator followed by the class name.

**instance method**

Any method that is invoked with respect to an instance of a class. Also called simply a method . See also class method .

**instance variable**

Any item of data that is associated with a particular object. Each instance of a class has its own copy of the instance variables defined in the class. Also called a field . See also class variable .

**instanceof**

A two-argument Java keyword that tests whether the runtime type of its first argument is assignment compatible with its second argument.

**int**

A Java keyword used to define a variable of type integer.

**interface**

A Java keyword used to define a collection of method definitions and constant values. It can later be implemented by classes that define this interface with the "implements" keyword.

I**nternet**

An enormous network consisting of literally millions of hosts from many organizations and countries around the world. It is physically put together from many smaller networks and data travels by a common set of protocols.

**IP**

Internet Protocol. The basic protocol of the Internet. It enables the unreliable delivery of individual packets from one host to another. It makes no guarantees about whether or not the packet will be delivered, how long it will take, or if multiple packets will arrive in the order they were sent. Protocols built on top of this add the notions of connection and reliability. See also TCP/IP .

**interpreter**

A module that alternately decodes and executes every statement in some body of code. The Java interpreter decodes and executes bytecode for the Java virtual machine 1 . See also compiler , r

**For J:**

**JAR**

JAR (Java Archive) is a platform-independent file format that aggregates many files into one. Multiple applets written in the Java programming language, and their requisite components (.class files, images, sounds and other resource files) can be bundled in a JAR file and subsequently downloaded to a browser in a single HTTP transaction. It also supports file compression and digital signatures.

**Java**

Sun's trademark for a set of technologies for creating and safely running software programs in both stand-alone and networked environments.

**Java 2 Platform**

The second generation of the Java platform. (The first generation was the JDK.) Also see "Java Platform" and "Java Platform Editions".

**Java Compatibility Kit (JCK)**

A test suite, a set of tools, and other requirements used to certify a Java platform implementation conformant both to the applicable Java platform specifications and to Java Software reference implementations.

**Java Database Connectivity (JDBC)**

An industry standard for database-independent connectivity between the Java platform and a wide range of databases. The JDBC provides a call-level API for SQL-based database access.

**Java Development Kit (JDK)**

A software development environment for writing applets and applications in the Java programming language. Technically, the JDK is the correct name for all versions of the Java platform from 1.0 to 1.1.x.

**Java Foundation Classes (JFC)**

An extension that adds graphical user interface class libraries to the Abstract Windowing Toolkit (AWT).

Java Runtime Environment (JRE)

A subset of the Java Development Kit (JDK) for end-users and developers who want to redistribute the runtime environment alone. The Java runtime environment consists of the Java virtual machine 1 , the Java core classes, and supporting files.

**Java virtual machine** 1

A software "execution engine" that safely and compatibly executes the byte codes in Java class files on a microprocessor (whether in a computer or in another electronic device).

**JavaBeans**

A portable, platform-independent reusable component model. A component that conforms to this model is called a bean.

JavaScript

A Web scripting language that is used in both browsers and Web servers. Like all scripting languages, it is used primarily to tie other components together or to accept user input.

**JavaSpaces**

A technology that provides distributed persistence and data exchange mechanisms for code in Java.

**JDBC**

See Java Database Connectivity.

**JDK**

Java Development Kit. A software development environment for writing applets and application in Java .

**JFC**

See Java Foundation Classes.

**Jini Technology**

A set of Java APIs that may be incorporated an optional package for any Java 2 Platform Edition. The Jini APIs enable transparent networking of devices and services and eliminates the need for system or network administration intervention by a user.

The Jini technology is currently an optional package available on all Java platform editions.

**JNDI**

See Java Naming and Directory Interface.

**JNI**

See Java Native Interface.

**JRE**

See Java Runtime Environment

**Just-in-time (JIT) Compiler**

A compiler that converts all of the bytecode into native machine code just as a Java program is run. This results in run-time speed improvements over code that is interpreted by a Java virtual machine.

**JVM**

See Java Virtual Machine (JVM).

**For K:**

**keyword**

Java sets aside words as keywords - these words are reserved by the language itself and therefore are not available as names for variables or methods.

**For M**

**member**

A field or method of a class. Unless specified otherwise, a member is not static.

**method**

A function defined in a class. See also instance method , class method . Unless specified otherwise, a method is not static.

**multithreaded**

Describes a program that is designed to have parts of its code execute concurrently. See also thread

**For O: object**

The principal building blocks of object-oriented programs. Each object is a programming unit consisting of data ( instance variables ) and functionality ( instance methods ). See also class .

**object-oriented design**

A software design method that models the characteristics of abstract or real objects using classes and objects.

octal

The numbering system using 8 as its base, using the numerals 0-7 as its digits. In programs written in the Java programming language, octal numbers must be preceded with 0. See also hexadecimal .

**optional packages**

The set or sets of APIs in a Java platform edition which are available with and may be supported in a compatible implementation.

Over time, optional packages may become required in an edition as the marketplace requires them. For P: package

A group of types . Packages are declared with the package keyword.

**peer**

In networking, any functional unit in the same layer as another entity.

**persistence**

The protocol for transferring the state of a bean between its instance variables and an underlying database.

**pixel**

The picture element on a display area, such as a monitor screen or printed page. Each pixel is individually accessible.

**POA**

Portable Object Adapter. A CORBA standard for building server-side applications that are portable across heterogeneous ORBs.

**primary key**

An object that uniquely identifies an entity bean within a home.

**primitive type**

A variable data type in which the variable's value is of the appropriate size and format for its type: a number, a character, or a boolean value.

**principal**

The identity assigned to an entity as a result of authentication.

**private**

A Java keyword used in a method or variable declaration. It signifies that the method or variable can only be accessed by other elements of its class.

**protected**

A Java keyword used in a method or variable declaration. It signifies that the method or variable can only be accessed by elements residing in its class, subclasses, or classes in the same package.

**Public**

A Java keyword used in a method or variable declaration. It signifies that the method or variable can be accessed by elements residing in other classes.

**For R: raster**

A two-dimensional rectangular grid of pixels.

**realm**

See security policy domain. Also, a string, passed as part of an HTTP request during basic authentication, that defines a protection space. The protected resources on a server can be partitioned into a set of protection spaces, each with its own authentication scheme and/or authorization database.

**reference**

A variable data type in which the variable's value is an address.

return

A Java keyword used to finish the execution of a method. It can be followed by a value required by the method definition.

**RMI**

See Java Remote Method Invocation .

**rollback**

The point in a transaction when all updates to any databases involved in the transaction are reversed.

**root**

In a hierarchy of items, the one item from which all other items are descended. The root item has nothing above it in the hierarchy. See also hierarchy , class , package .

**For S: sandbox**

Comprises a number of cooperating system components, ranging from security managers that execute as part of the application, to security measures designed into the Java virtual machine 1 and the language itself. The sandbox protects sensitive system information from access by untrusted and potentially malicious applications.

**scope**

A characteristic of an identifier that determines where the identifier can be used. Most identifiers in the Java programming environment have either class or local scope. Instance and class variables and methods have class scope; they can be used outside the class and its subclasses only by prefixing them with an instance of the class or (for class variables and methods) with the class name. All other variables are declared within methods and have local scope; they can be used only within the enclosing block.

**short**

A Java keyword used to define a variable of type short.

**single precision**

In the Java language specification, describes a floating point number with 32 bits of data. See also double precision .

**static field**

Another name for class variable .

**static method**

Another name for class method .

**stream**

A stream is simply a byte-stream of data that is sent from a sender to a receiver. There are two basic categories, so the java.io package includes two abstract classes ( InputStream and OutputStream).

**subarray**

An array that is inside another array.

**subclass**

A class that is derived from a particular class, perhaps with one or more classes in between. See also superclass , supertype .

**subtype**

If type X extends or implements type Y, then X is a subtype of Y. See also supertype .

**superclass**

A class from which a particular class is derived, perhaps with one or more classes in between. See also subclass , subtype .

**super**

A Java keyword used to access members of a class inherited by the class in which it appears.

**supertype**

The supertypes of a type are all the interfaces and classes that are extended or implemented by that type. See also subtype , superclass .

**switch**

A Java keyword used to evaluate a variable that can later be matched with a value specified by the case keyword in order to execute a group of statements.

**Swing**

A collection of graphical user interface (GUI) components that runs uniformly on any native platform which supports the Java virtual machine \*. Because they are written entirely in the Java programming language, these components may provide functionality above and beyond that provided by native-platform equivalents. (Contrast with AWT.)

**REFERENCES**

**1.HOW TO MAKE A TESTFIELD THAT ACCEPTS ONLY NUMBER:** [www.youtube.com/watch?v=9B5ZgItFxNA](https://www.youtube.com/watch?v=9B5ZgItFxNA)

**2. How to put a thumbnail icon onto the application:**  <https://stackoverflow.com/questions/26615889/how-to-change-the-launcher-logo-of-an-app-in-android-studio>

**3.** [**Package Installer has stopped**](https://stackoverflow.com/questions/9367832/package-installer-has-stopped)**:** [**https://www.youtube.com/watch?v=Fglc2VgdnbI**](https://www.youtube.com/watch?v=Fglc2VgdnbI)

**4. JavaFX Ubuntu Installation:**

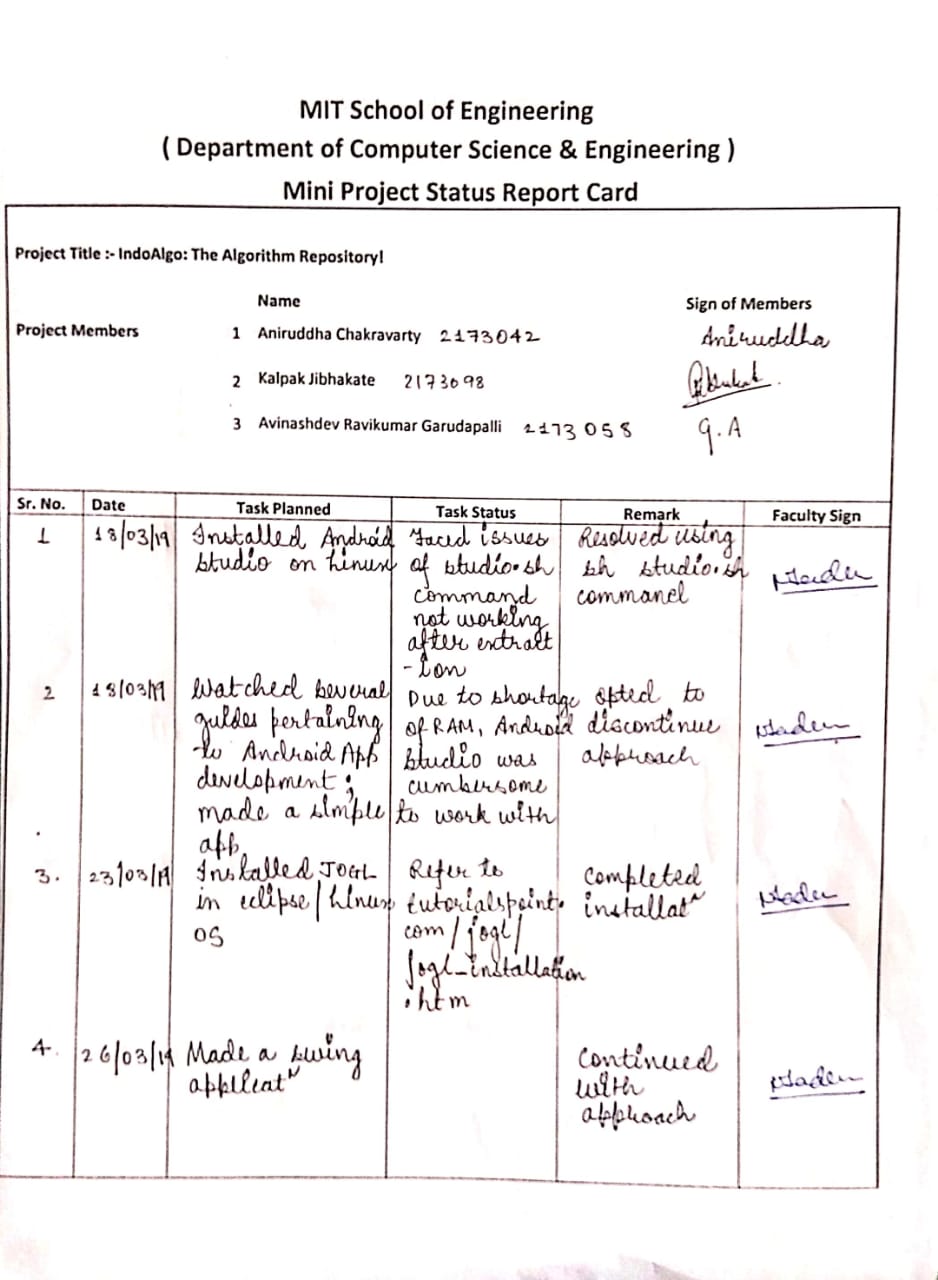
Refer to <https://github.com/buckyroberts/Source-Code-from-Tutorials/tree/master/JavaFX/>

for code format and the command is: sudo apt install openjdk-8-jdk openjfx

**5. For Java OpenGL installation refer:** <https://www.tutorialspoint.com/jogl/jogl_installation.htm/>

**6. For opengl menu refer :**

<https://github.com/nifty-gui/nifty-gui/wiki/Working-from-Source/>

**ANNEXURE A**

