

**M.Sc C.S - II**

**SEM III**

**Journal**

|  |  |
| --- | --- |
| **Roll No.** | 4701 |
| **Name** | Abhang Mane |
| **Subject** | Ubiqutous Computing |



**CERTIFICATE**

This is here to certify that **Mr.** **Abhang Sanjay Mane** Seat Number **4701** of M.Sc-II Computer Science, has satisfactorily completed the required number of Practicals prescribed by Thakur College Of Science And Commerce during the academic year 2023 – 2024.

Date:

Place: Kandivali,Mumbai

**Ms.Riddhi Mishra**

**Teacher In-Charge Head of Department**

**External Examiner**

**INDEX**

| **PrNo** | **Aim** | **Date** | **Sign** |
| --- | --- | --- | --- |
| 1 | Design and develop location based messaging app | 15/07/23 |  |
| 2 | Design and develop chat messaging app which is a location-based | 22/07/23 |  |
| 3 | Demonstrate use of OpenGTS (Open Source GPS Tracking System) | 17/08/23 |  |
| 4 | Develop application demonstrating Human Computer Interaction  (Using KeyListener) | 26/08/23 |  |
| 5 | Develop application demonstrating Human Computer Interaction  (Using MouseListener) | 02/09/23 |  |
| 6 | Write a Java Card applet | 16/09/23 |  |

**Practical 1**

**Aim:**Design and develop a location based messaging app

**Theory:**  
Location Manager is a crucial component in mobile application development that allows developers to access and retrieve the geographical coordinates of a device's current location, including longitude and latitude. It is particularly useful for applications that require location-based services, such as mapping, navigation, and location-aware content delivery. In this one-page theory, we will explore Location Manager, its key functions, and how it facilitates the retrieval of longitude and latitude information.

Location Manager

Location Manager is an integral part of the Android and iOS platforms, providing a high-level interface for obtaining location-related data from various sources, such as GPS, Wi-Fi, and cellular networks. Its primary functions include:

Location Updates: Location Manager allows developers to request periodic updates on the device's location. These updates can be based on time intervals or distance traveled, ensuring that applications stay up-to-date with the user's location.

Provider Selection: Location Manager supports multiple location providers, such as GPS, Network, and Passive. Developers can choose the most suitable provider based on their accuracy and power consumption requirements.

Permissions: Location Manager enforces user permissions for accessing location data. Developers must request appropriate permissions from users to access their location, ensuring privacy and security.

Listener Callbacks: Developers can register LocationListeners to receive updates when the device's location changes. This allows applications to respond dynamically to location changes and provide location-based services.

Geocoding and Reverse Geocoding: Location Manager can also perform geocoding (converting addresses into coordinates) and reverse geocoding (converting coordinates into addresses). This feature is valuable for mapping and location-based search functionalities.

Longitude and Latitude

Longitude and latitude are essential components of the geographical coordinate system, which represents any point on the Earth's surface. Here's a brief explanation of these terms:

Longitude: Longitude measures the east-west position of a point on Earth. It is represented in degrees, with values ranging from -180° (West) to +180° (East). The Prime Meridian, located at 0° longitude, passes through Greenwich, London.

Latitude: Latitude measures the north-south position of a point on Earth. It is also represented in degrees, with values ranging from -90° (South) to +90° (North). The Equator is at 0° latitude, dividing the Earth into Northern and Southern Hemispheres.

When using Location Manager, longitude and latitude are typically retrieved as part of a Location object, which includes additional information such as altitude, speed, and accuracy.

In summary, Location Manager is a vital component in mobile app development, enabling developers to access and utilize location-based information, including longitude and latitude. By integrating Location Manager into their apps, developers can create location-aware applications that offer a wide range of services and functionality, from navigation and mapping to location-based recommendations and social networking features.

**Source Code:**

**MainActivity.java;**

package com.example.mycurrentlocation;  
  
import android.Manifest;  
import android.annotation.SuppressLint;  
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.view.View;  
import android.widget.Button;  
import android.widget.TextView;  
import android.widget.Toast;  
  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.core.app.ActivityCompat;  
import androidx.core.content.ContextCompat;  
  
import java.util.List;  
import java.util.Locale;  
  
public class MainActivity extends AppCompatActivity implements LocationListener {  
  
 Button button\_location;  
 TextView textView\_location;  
 LocationManager locationManager;  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
  
 textView\_location = findViewById(R.id.text\_location);  
 button\_location = findViewById(R.id.button\_location);  
 //Runtime permissions  
 if (ContextCompat.checkSelfPermission(MainActivity.this, Manifest.permission.ACCESS\_FINE\_LOCATION)  
 != PackageManager.PERMISSION\_GRANTED){  
 ActivityCompat.requestPermissions(MainActivity.this,new String[]{  
 Manifest.permission.ACCESS\_FINE\_LOCATION  
 },100);  
 }  
  
  
 button\_location.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 //create method  
 getLocation();  
 }  
 });  
  
  
  
 }  
  
 @SuppressLint("MissingPermission")  
 private void getLocation() {  
  
 try {  
 locationManager = (LocationManager) getApplicationContext().getSystemService(LOCATION\_SERVICE);  
 locationManager.requestLocationUpdates(LocationManager.GPS\_PROVIDER,5000,5,MainActivity.this);  
  
 }catch (Exception e){  
 e.printStackTrace();  
 }  
  
 }  
  
 @Override  
 public void onLocationChanged(Location location) {  
 textView\_location.setText("" + location.getLatitude() + "," + location.getLongitude());  
  
 }  
  
 @Override  
 public void onStatusChanged(String provider, int status, Bundle extras) {  
  
 }  
  
 @Override  
 public void onProviderEnabled(String provider) {  
  
 }  
  
 @Override  
 public void onProviderDisabled(String provider) {  
  
 }  
}

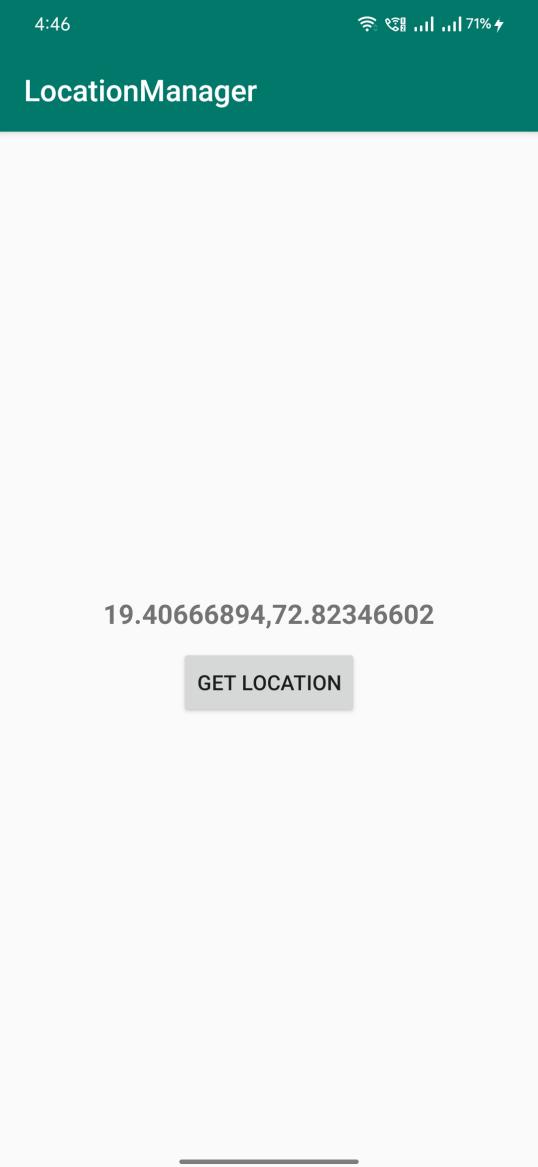
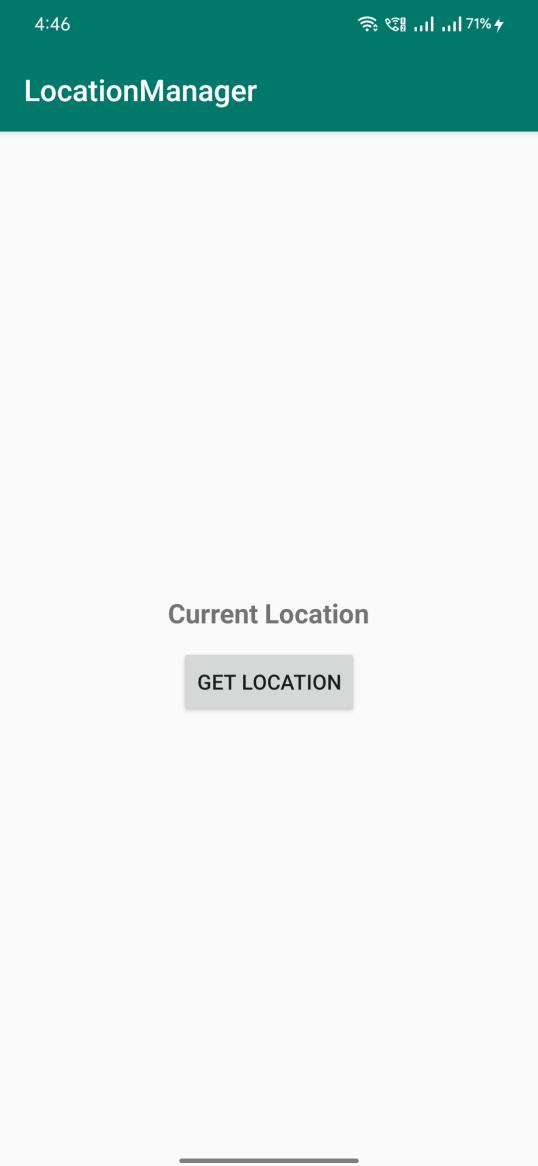
**AndroidManifest.xml:**

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android">  
  
 <uses-permission android:name="android.permission.INTERNET"/>  
 <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION"/>  
  
 <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>

**activity\_main.xml:**

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:orientation="vertical"  
 android:layout\_gravity="center"  
 android:gravity="center"  
 android:layout\_height="match\_parent"  
 tools:context=".MainActivity">  
   
 <TextView  
 android:id="@+id/text\_location"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_margin="10dp"  
 android:text="Current Location"  
 android:textSize="18sp"  
 android:textStyle="bold" />  
  
 <Button  
 android:id="@+id/button\_location"  
 android:text="Get Location"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"/>  
  
</LinearLayout>

**Output:**



**Conclusion:**Hence a location based messaging app was designed and developed successfully.

**Practical 2**

**Aim:**Design and develop chat messaging app which is a location-based

**Theory:**

A location-based messaging app seeks to connect users who are in the same

general vicinity. This is done by using GPS to show the location pins of other

users on a map, making it an ideal feature for meeting new people while on

vacation or during a concert or event

**Source Code:**

**MainActivity.java:**

package com.example.prac2;  
  
import android.Manifest;  
import android.content.Context;  
import android.content.pm.PackageManager;  
import android.location.Location;  
import android.location.LocationListener;  
import android.location.LocationManager;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.View;  
import android.widget.Button;  
import android.widget.TextView;  
  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.core.app.ActivityCompat;  
  
public class  
MainActivity extends AppCompatActivity implements LocationListener  
{  
 TextView t1, t2, t3;  
 Button b1;  
 protected LocationManager locationManager;  
 protected LocationListener locationListener;  
 double lat,longg;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 t1 = (TextView) findViewById(R.id.textView);  
 t2 = (TextView) findViewById(R.id.textView3);  
 t3 = (TextView) findViewById(R.id.textView2);  
 b1 = (Button) findViewById(R.id.button);  
 b1.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View view) {  
 t1.setText("Latitude = " + lat);  
 t2.setText("Longitude = "+ longg);  
 if(lat<38 &&lat>36 &&longg<122 &&longg>118)  
 {  
 t3.setText("In-Side The Area");  
 }  
 else  
 {  
 t3.setText("Out-Side The Area");  
 }  
 }  
 });  
 locationManager= (LocationManager)  
 getApplicationContext().getSystemService(LOCATION\_SERVICE);  
 if (ActivityCompat.checkSelfPermission(this,  
 Manifest.permission.ACCESS\_FINE\_LOCATION) !=  
 PackageManager.PERMISSION\_GRANTED && ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS\_COARSE\_LOCATION) !=  
 PackageManager.PERMISSION\_GRANTED) {  
 return;  
 }  
 locationManager.requestLocationUpdates(LocationManager.GPS\_PROVIDER, 0, 0, this);  
 }  
 @Override  
 public void onLocationChanged(Location location) {  
// txtLat = (TextView) findViewById(R.id.textview1);  
 lat=location.getLatitude();  
 longg= location.getLongitude();  
 Log.d(""+lat,""+lat);  
 Log.d(""+longg,""+longg);  
 if(lat<38 &&lat>36 &&longg<122 &&longg>118)  
 {  
 t3.setText("In-Side The Area");  
 }  
 else  
 {  
 t3.setText("Out-Side The Area");  
 }  
//t1.setText("Latitude:" + location.getLatitude() + ",Longitude:" + location.getLongitude());  
 }  
 @Override  
 public void onProviderDisabled(String provider) {  
 Log.d("Latitude","disable");  
 }  
 @Override  
 public void onProviderEnabled(String provider) {  
 Log.d("Latitude","enable");  
 }  
  
 @Override  
 public void onStatusChanged(String provider, int status, Bundle  
 extras) {  
 Log.d("Latitude","status");  
 }  
}

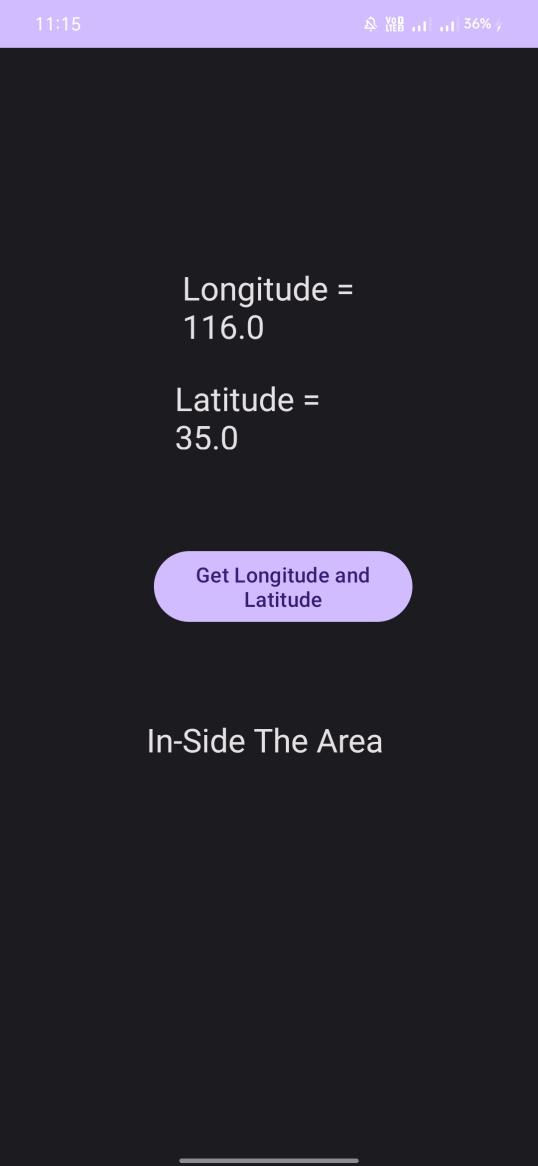
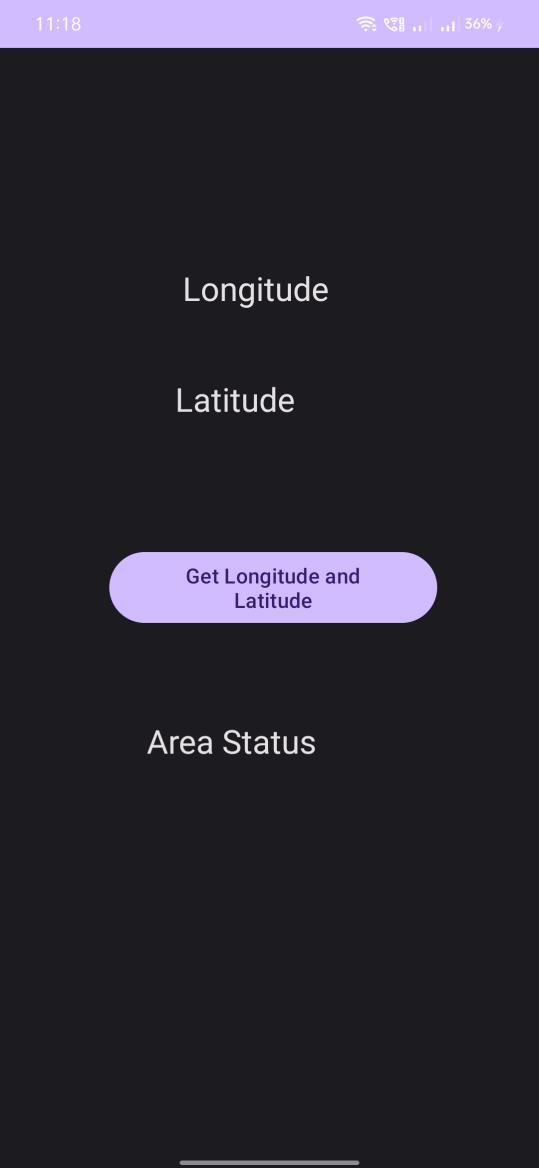
**activity\_main.xml:**

<?xml version="1.0" encoding="utf-8"?>  
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:paddingBottom="@dimen/activity\_vertical\_margin"  
 android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"  
 tools:context="com.example.prac2.MainActivity">  
  
 <TextView  
 android:id="@+id/textView"  
 android:layout\_width="172dp"  
 android:layout\_height="49dp"  
 android:layout\_above="@+id/button"  
 android:layout\_alignParentStart="true"  
 android:layout\_alignParentLeft="true"  
 android:layout\_alignParentEnd="true"  
 android:layout\_alignParentRight="true"  
 android:layout\_marginStart="117dp"  
 android:layout\_marginLeft="117dp"  
 android:layout\_marginEnd="122dp"  
 android:layout\_marginRight="122dp"  
 android:layout\_marginBottom="64dp"  
 android:text="Latitude"  
 android:textAppearance="?android:attr/textAppearanceLarge" />  
  
 <TextView  
 android:id="@+id/textView3"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="57dp"  
 android:layout\_above="@+id/textView"  
 android:layout\_alignParentStart="true"  
 android:layout\_alignParentLeft="true"  
 android:layout\_alignParentEnd="true"  
 android:layout\_alignParentRight="true"  
 android:layout\_marginStart="122dp"  
 android:layout\_marginLeft="122dp"  
 android:layout\_marginEnd="120dp"  
 android:layout\_marginRight="120dp"  
 android:layout\_marginBottom="17dp"  
 android:text="Longitude"  
 android:textAppearance="?android:attr/textAppearanceLarge" />  
  
 <Button  
 android:id="@+id/button"  
 android:layout\_width="270dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_alignParentStart="true"  
 android:layout\_alignParentLeft="true"  
 android:layout\_alignParentEnd="true"  
 android:layout\_alignParentRight="true"  
 android:layout\_alignParentBottom="true"  
 android:layout\_marginStart="73dp"  
 android:layout\_marginLeft="73dp"  
 android:layout\_marginEnd="68dp"  
 android:layout\_marginRight="68dp"  
 android:layout\_marginBottom="360dp"  
 android:text="Get Longitude and Latitude" />  
  
 <TextView  
 android:id="@+id/textView2"  
 android:layout\_width="221dp"  
 android:layout\_height="69dp"  
 android:layout\_alignParentStart="true"  
 android:layout\_alignParentLeft="true"  
 android:layout\_alignParentEnd="true"  
 android:layout\_alignParentRight="true"  
 android:layout\_alignParentBottom="true"  
 android:layout\_marginStart="98dp"  
 android:layout\_marginLeft="98dp"  
 android:layout\_marginEnd="92dp"  
 android:layout\_marginRight="92dp"  
 android:layout\_marginBottom="231dp"  
 android:text="Area Status"  
 android:textAppearance="?android:attr/textAppearanceLarge" />  
</RelativeLayout>

**AndroidManifest.xml:**

<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools">  
  
 <uses-permission android:name="android.permission.INTERNET"/>  
 <uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION"/>  
 <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />  
  
 <application  
 android:allowBackup="true"  
 android:dataExtractionRules="@xml/data\_extraction\_rules"  
 android:fullBackupContent="@xml/backup\_rules"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/Theme.Prac2"  
 tools:targetApi="31">  
 <activity  
 android:name=".MainActivity"  
 android:exported="true">  
 <intent-filter>  
 <action android:name="android.intent.action.MAIN" />  
  
 <category android:name="android.intent.category.LAUNCHER" />  
 </intent-filter>  
 </activity>  
 </application>  
  
</manifest>

**Output:**



**Conclusion:**Hence a chat messaging app which is location-based was developed and designed successfully.

**Practical 3**

**Aim**:Demonstrate use of OpenGTS (Open Source GPS Tracking System)

**Theory:**

OpenGTS (Open GPS Tracking System) is an open-source software platform designed for tracking and managing the location of vehicles, assets, and personnel using GPS technology. It provides a comprehensive framework for real-time GPS tracking, fleet management, and location-based services. Below is a one-page overview of OpenGTS.

Introduction to OpenGTS: OpenGTS is an open-source GPS tracking platform that enables organizations to monitor the location and movement of vehicles, assets, and personnel in real-time. It is a highly flexible and customizable solution, making it suitable for various industries, including logistics, transportation, field services, and personal tracking.

Key Features of OpenGTS:

Real-Time GPS Tracking: OpenGTS supports real-time tracking of vehicles and assets equipped with GPS devices. It continuously collects and updates location data, allowing users to monitor positions on a map.

Geofencing: Users can define virtual geographical boundaries, known as geofences, and receive notifications when tracked objects enter or exit these predefined areas. Geofencing is valuable for security and alerting purposes.

Event Management: OpenGTS can trigger events and notifications based on predefined criteria. This includes events such as speeding, idling, or unauthorized vehicle use. Notifications can be sent via email, SMS, or other communication channels.

Reports and History: The platform offers a wide range of reporting options, allowing users to access historical tracking data, generate trip reports, and analyze vehicle performance. Reports can be customized to meet specific requirements.

Multiple Device Support: OpenGTS is device-agnostic and supports a variety of GPS tracking devices, making it versatile and adaptable to different hardware setups.

User Access Control: Role-based access control ensures that only authorized users can access sensitive tracking data and features.

Integration: OpenGTS provides APIs and tools for integration with other systems, enabling organizations to connect GPS tracking with their existing software and databases.

Architecture of OpenGTS:

OpenGTS follows a client-server architecture, where GPS tracking devices (clients) send location data to a central server. The server processes and stores this data, making it accessible to users through a web-based interface or APIs. Key components include:

Data Accumulation Server: Collects, validates, and stores incoming GPS data from tracking devices.

Web-Based Interface: Allows users to access tracking information, reports, and configure geofences and alerts.

Database: Stores historical tracking data, user settings, and system configurations.

APIs: Provide programmatic access to the platform's functionality, facilitating integration with external systems.

**Settings:-**

**Required Software:-**

**1) JDK 1.6**

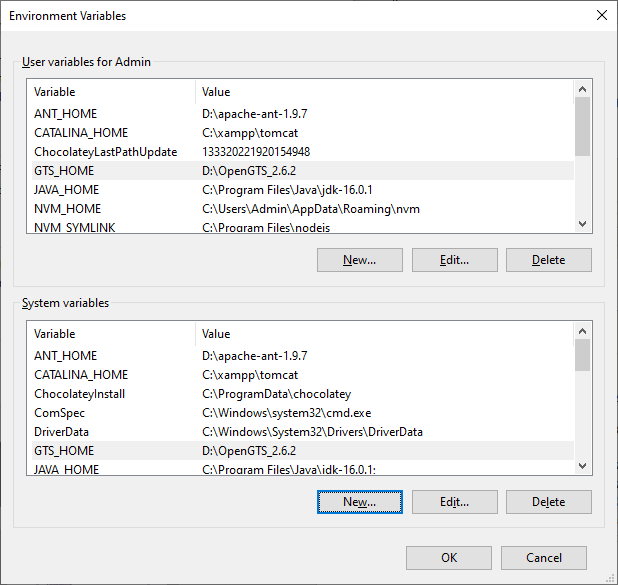
**2) XAMPP Server**

**3) Mysql-java connector**

**4) OpenGTS application.**

http://www.opengts.org/

**Set Environment Variables:**



**5) Open command Prompt and go to D:\OpenGTS\_2.6.2**

**6) Type command ant all**

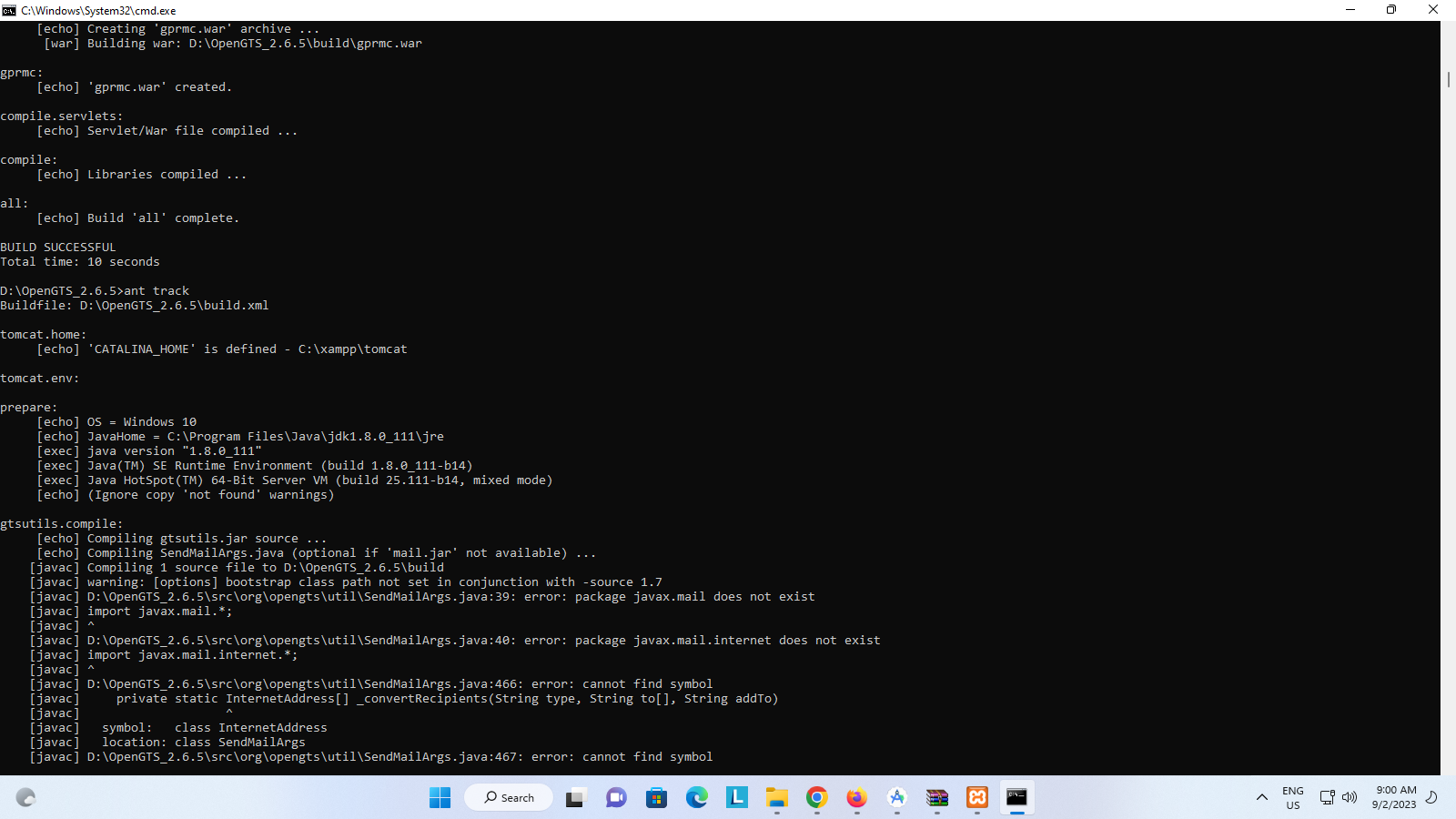
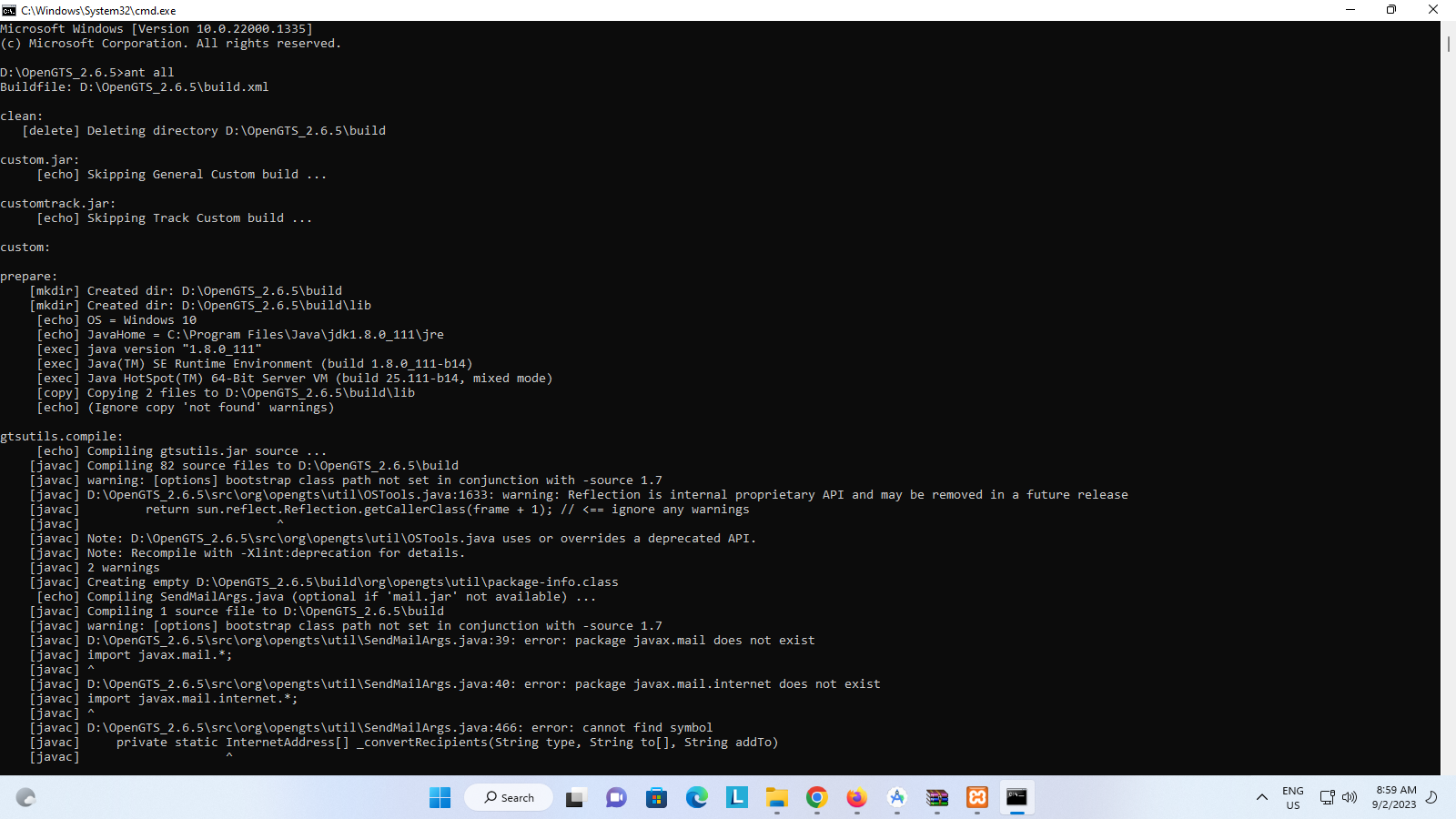
**7) Type command ant track**

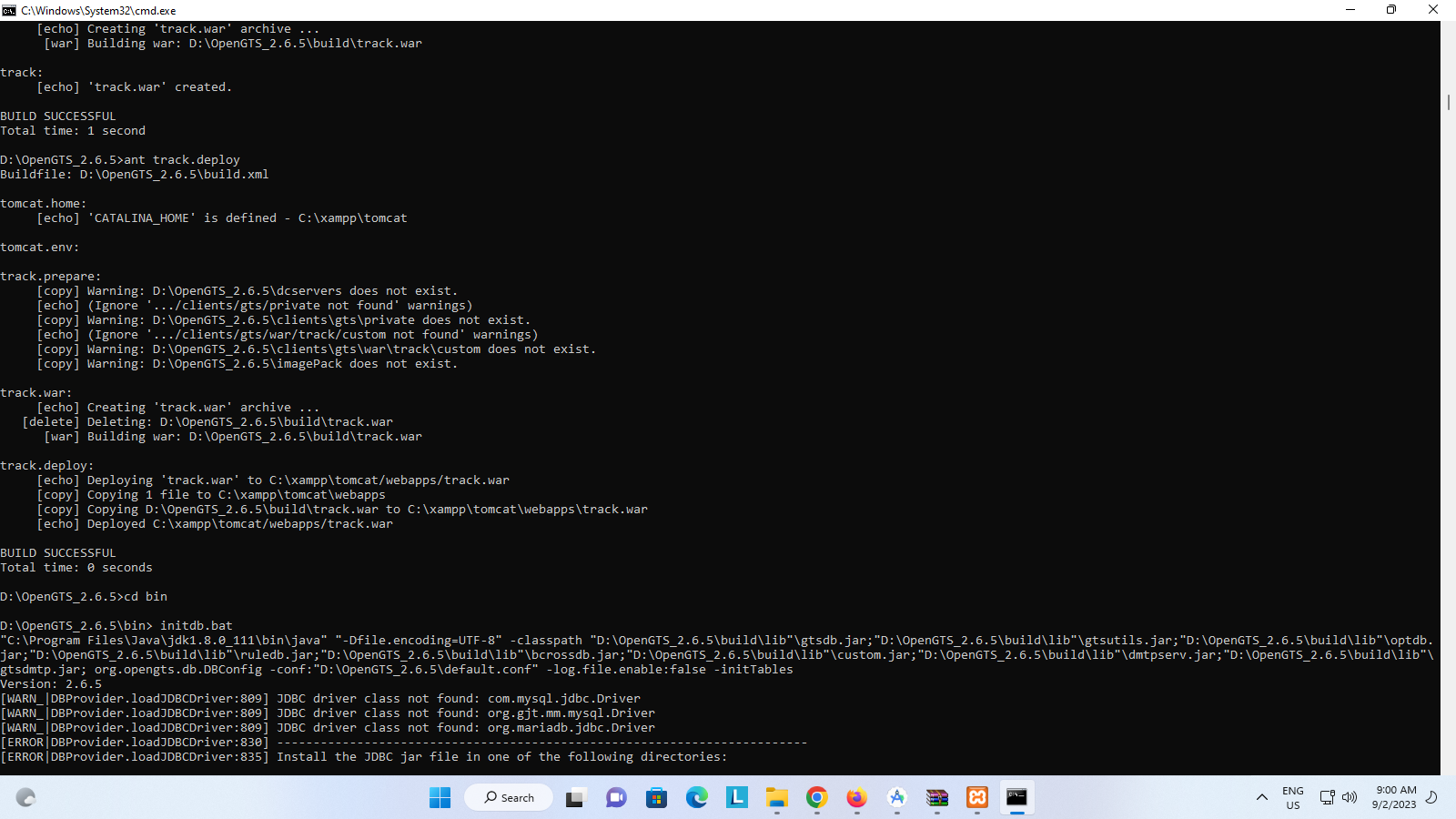
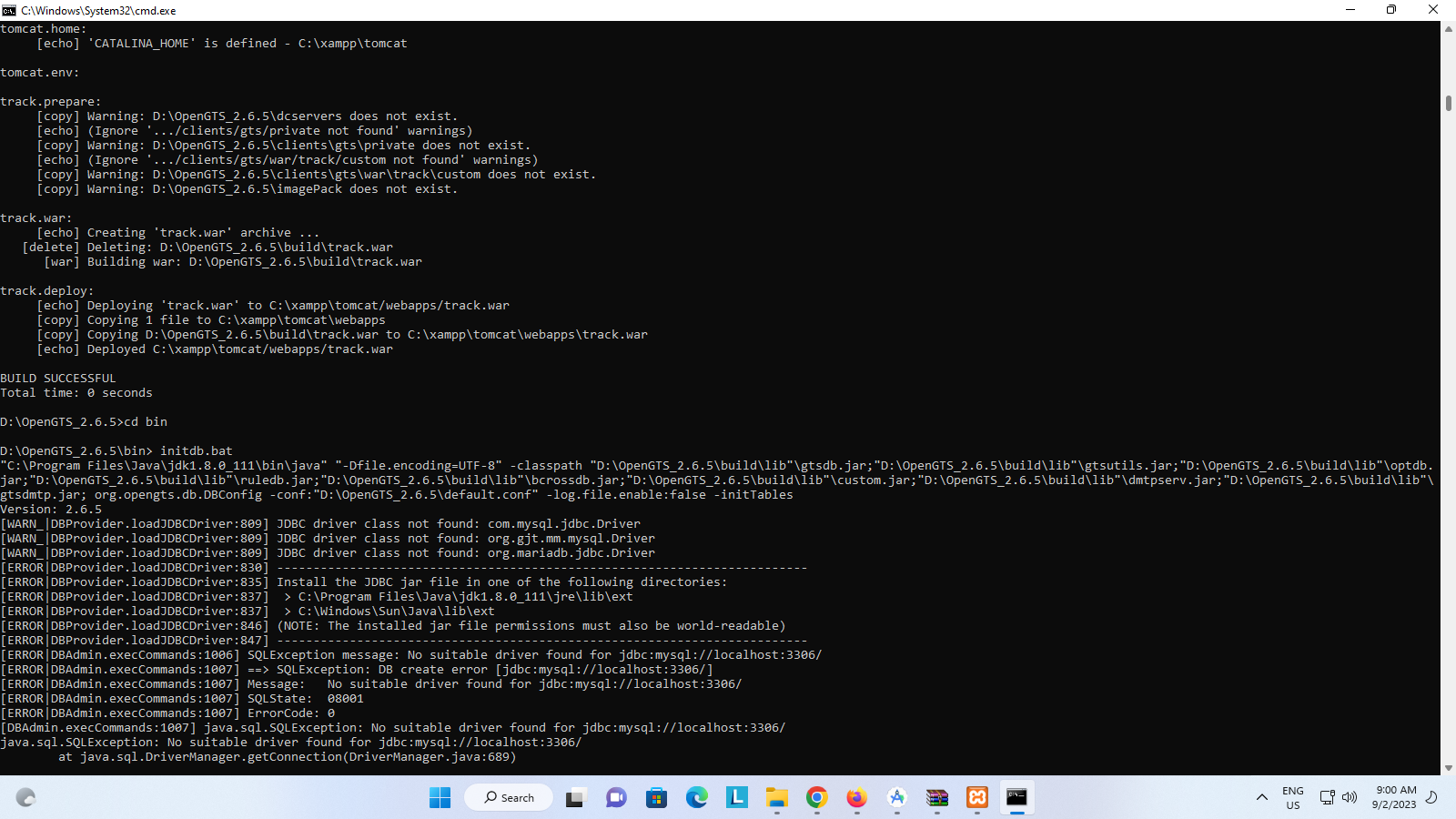
**8) Type command ant track.deploy**

**9) Go to D:\OpenGTS\_2.6.2\bin ; Type command initdb.bat**

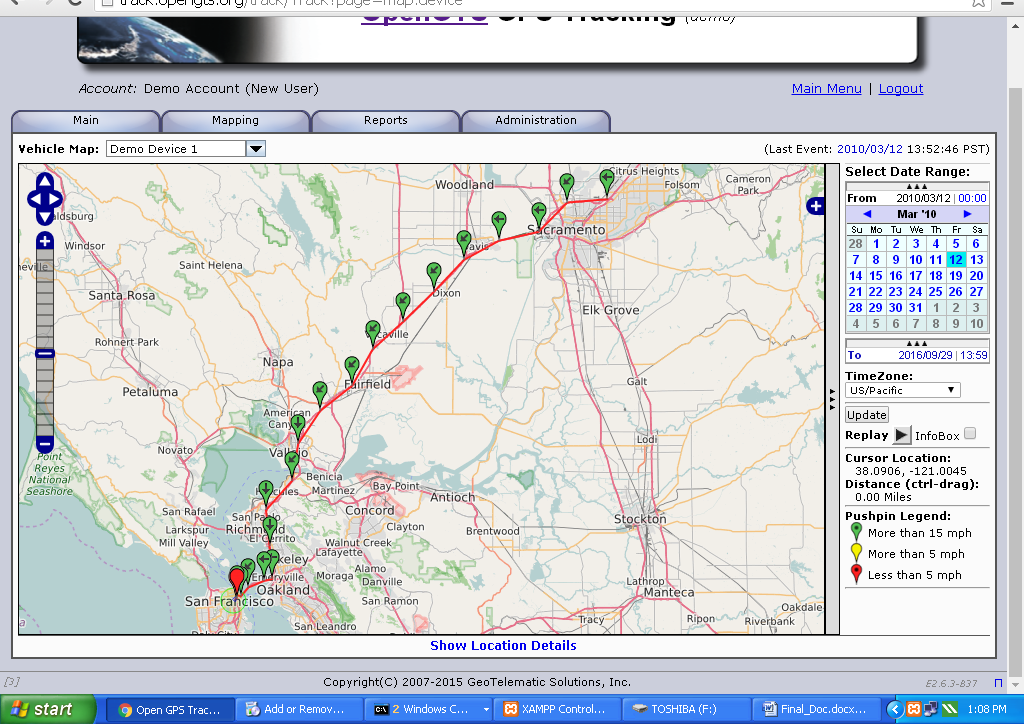
**10) Type Command admin.bat Account –account:admin –pass:123456 –create**

**11) Type url localhost:8080/track/Track and login with admin and 123456**

****

****





**Conclusion:**Hence use of OpenGTS (Open Source GPS Tracking System) was demonstrated successfully.

**Practical 4**

**Aim:**Develop application demonstrating Human Computer Interaction

(Using KeyListener)

**Theory:**

Human-Computer Interaction (HCI) Using Key Listeners in Ubiquitous Computing

Introduction to Ubiquitous Computing: Ubiquitous computing, also known as pervasive computing, is a paradigm where computing technology is seamlessly integrated into our everyday environment, making it an integral part of our lives. In this context, Human-Computer Interaction (HCI) plays a crucial role in enabling users to interact with and control the myriad of computing devices and systems that surround them.

Key Listener in Ubiquitous Computing: A key listener is a fundamental component of HCI in ubiquitous computing. It refers to software or hardware mechanisms that capture and interpret user input from physical or virtual keyboards. Key listeners enable users to interact with computing devices in their environment, such as smart appliances, wearable devices, and IoT (Internet of Things) gadgets, by translating physical keystrokes into meaningful actions or commands.

Key Concepts in HCI Using Key Listeners in Ubiquitous Computing:

User-Centric Design: In ubiquitous computing, the design of key listeners and the overall HCI experience should prioritize user needs and preferences. Understanding user contexts and behaviors is essential for creating efficient and intuitive interfaces.

Context Awareness: Ubiquitous computing environments are context-aware, meaning they can adapt to the user's location, preferences, and current activities. Key listeners are programmed to respond to user input in contextually relevant ways. For example, adjusting the lighting in a room or controlling a home entertainment system.

Multi-Modal Interaction: In addition to traditional keyboard input, ubiquitous computing leverages various input modalities, including touchscreens, voice recognition, and gesture controls. Key listeners may need to support multiple input sources and seamlessly switch between them for a cohesive user experience.

Feedback and Confirmation: Effective HCI using key listeners provides feedback to users to confirm that their input has been received and understood. Visual, auditory, or haptic feedback can enhance user confidence and prevent errors.

Privacy and Security: Given the pervasive nature of computing in everyday life, key listeners must consider privacy and security concerns. User data should be handled securely, and users must have control over the extent to which devices listen and respond to their input.

Interoperability: Ubiquitous computing environments often involve heterogeneous devices and platforms. Key listeners may need to interface with various systems and ensure interoperability between them to create a seamless user experience.

Applications of Key Listeners in Ubiquitous Computing:

Smart Homes: Key listeners enable users to control lighting, heating, security systems, and entertainment devices with physical or virtual key inputs.

Wearable Devices: Wearables, such as smartwatches, often use key listeners to interpret user interactions with limited input interfaces.

IoT Devices: Key listeners play a role in managing and controlling a multitude of IoT devices, from thermostats to smart appliances.

Smart Cities: In a smart city context, key listeners might be used for interacting with public infrastructure like kiosks, streetlights, and transportation systems.

**Source Code:**

package keylistenerexample;

import java.awt.\*;

import java.awt.event.\*;

public class hci extends Frame implements KeyListener{

 Label l;

 TextArea area;

 hci(){

 l=new Label();

 l.setBounds(20,50,100,20);

 area=new TextArea();

 area.setBounds(20,80,300, 300);

 area.addKeyListener(this);

 add(l);add(area);

 setSize(400,400);

 setLayout(null);

 setVisible(true);

 setTitle("4701-Abhang Mane");

 }

 public void keyPressed(KeyEvent e) {

 l.setText("Key Pressed");

 }

 public void keyReleased(KeyEvent e) {

 l.setText("Key Released");

 }

 public void keyTyped(KeyEvent e) {

 l.setText("Key Typed");

 }

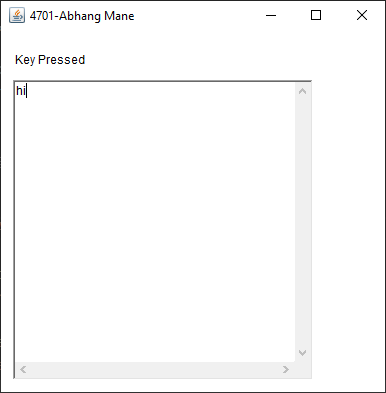
 public static void main(String[] args) {

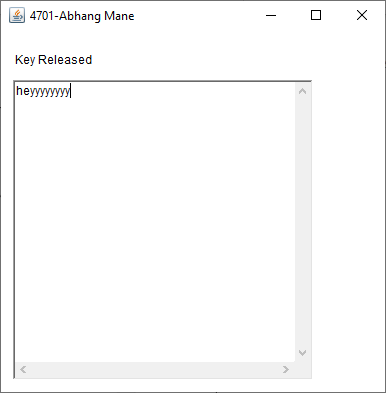
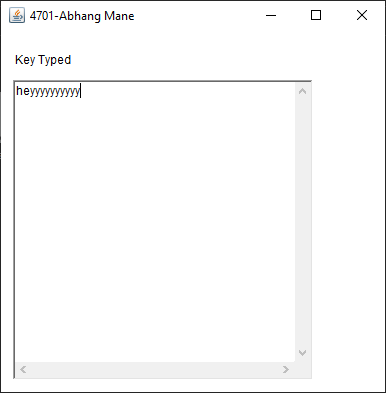
 new hci();

 }

}

**Output:**





**Conclusion:**Hence an application demonstrating Human Computer Interaction

(Using KeyListener) was created and developed successfully.

**Practical 5**

**Aim:**Develop application demonstrating Human Computer Interaction (Using MouseListener)

**Theory:**

Mouse Listener in Ubiquitous Computing: A mouse listener is a software component or mechanism that captures and interprets user input from a pointing device, such as a mouse or a touchpad. In ubiquitous computing, mouse listeners serve as a bridge between users and the interconnected computational systems present in their environment, enabling intuitive and precise interactions.

Key Concepts in HCI Using Mouse Listeners in Ubiquitous Computing:

Natural Interaction: Mouse listeners contribute to the naturalness of interaction in ubiquitous computing environments. Users can interact with digital elements embedded in the physical world by simply pointing and clicking, a familiar action from traditional desktop computing.

Context Awareness: Ubiquitous computing environments are context-aware, meaning they can adapt to the user's location, preferences, and activities. Mouse listeners are designed to respond contextually to user input, offering relevant options or actions based on the user's context.

Multi-Modal Interaction: Beyond mouse input, ubiquitous computing supports multiple interaction modalities, including touchscreens, voice commands, and gestures. Mouse listeners should seamlessly integrate with these modalities, allowing users to switch between them as needed.

Feedback and Responsiveness: Effective HCI using mouse listeners involves providing immediate feedback to users, such as visual cues or haptic responses, to confirm that their input has been registered and understood by the system. Responsiveness is critical for user satisfaction and preventing errors.

Privacy and Security: Given the pervasiveness of computing, privacy and security concerns are paramount. Mouse listeners must ensure secure handling of user data and provide users with control over the extent to which devices capture and interpret their input.

Interoperability: Ubiquitous computing environments often consist of diverse devices and platforms. Mouse listeners need to be compatible with various systems and promote interoperability to deliver a seamless user experience.

Applications of Mouse Listeners in Ubiquitous Computing:

Smart Environments: Mouse listeners are used in smart homes and offices, enabling users to control lighting, temperature, and multimedia systems with point-and-click interactions.

Wearable Devices: Wearable technology often employs mouse listeners for navigation and interaction with small screens or touch-sensitive surfaces.

**Source Code:**

import java.awt.\*;

import java.awt.event.\*;

public class MouseListenerExample extends Frame implements MouseListener {

    Label l;

    MouseListenerExample() {

        addMouseListener(this);

        l = new Label();

        l.setBounds(20, 50, 100, 20);

        add(l);

        setSize(300, 300);

        setLayout(null);

        setVisible(true);

    }

    public void mouseClicked(MouseEvent e) {

        l.setText("Mouse Clicked");

    }

    public void mouseEntered(MouseEvent e) {

        l.setText("Mouse Entered");

    }

    public void mouseExited(MouseEvent e) {

        l.setText("Mouse Exited");

    }

    public void mousePressed(MouseEvent e) {

        l.setText("Mouse Pressed");

    }

    public void mouseReleased(MouseEvent e) {

        l.setText("Mouse Released");

    }

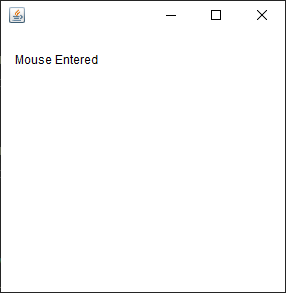
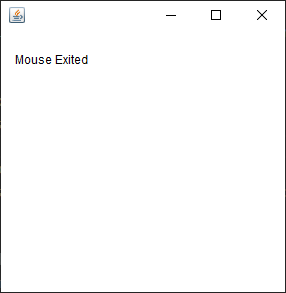
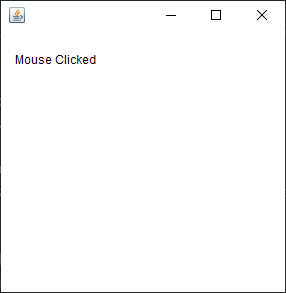
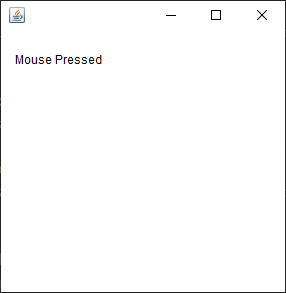
    public static void main(String[] args) {

        new MouseListenerExample();

    }

}

**Output:**

****

**Conclusion:**Hence an application demonstrating Human Computer Interaction (Using MouseListener) was developed successfully.

**Practical 6**

**Aim:**Write a Java Card applet

**Theory:**

Introduction to Java Card: Java Card is a subset of the Java platform designed for resource-constrained devices, particularly smart cards. It allows developers to create secure and portable applications, called applets, that can be executed on smart cards and similar embedded systems. These applets are written in the Java programming language, offering the benefits of platform independence, security, and ease of development.

Key Concepts and Features of Java Card:

Platform Independence: Java Card applications are written in Java, providing platform independence. This means an applet developed on one Java Card platform can be executed on another compatible platform without modification, enhancing portability.

Security: Security is paramount in Java Card. It employs various mechanisms to protect sensitive data and ensure the integrity of applications. Features like runtime isolation and secure channels are used to prevent unauthorized access.

Applet Model: Java Card applications are organized into applets, which are small, self-contained modules. Each applet has its own data and code, making it possible to load, delete, and manage applets independently on the smart card.

Limited Resources: Smart cards have limited resources, including CPU power, memory, and storage. Java Card applications are designed to be highly efficient to operate within these constraints.

Runtime Environment: Java Card includes a runtime environment specifically tailored for smart cards. It provides core libraries and services, including garbage collection and exception handling, to support applet execution.

Global Platform: Java Card is often used in conjunction with the GlobalPlatform standard, which defines a standardized way to manage and interact with applications on smart cards. GlobalPlatform enhances interoperability between cards and card management systems.

Applications of Java Card:

Payment Systems: Java Card technology is widely used in payment cards, such as credit and debit cards with EMV (Europay, Mastercard, and Visa) chips. These cards securely store and process financial data.

Authentication and Access Control: Smart cards equipped with Java Card technology are used for secure authentication and access control in various domains, including government ID cards and secure access cards.

**Steps:**

**Download Java Card SDK from here:**

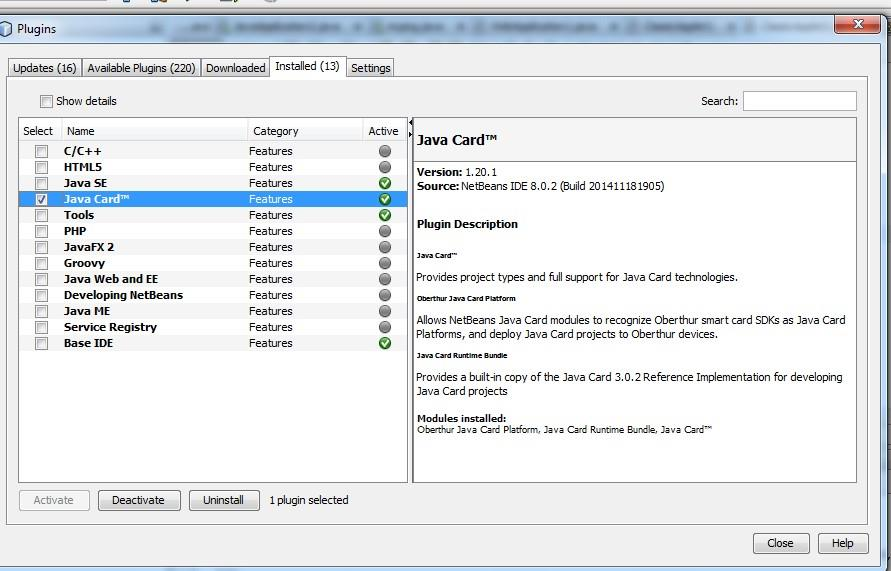
http://www.oracle.com/technetwork/java/embedded/javacard/downloads/jav

acard-sdk2043229.html

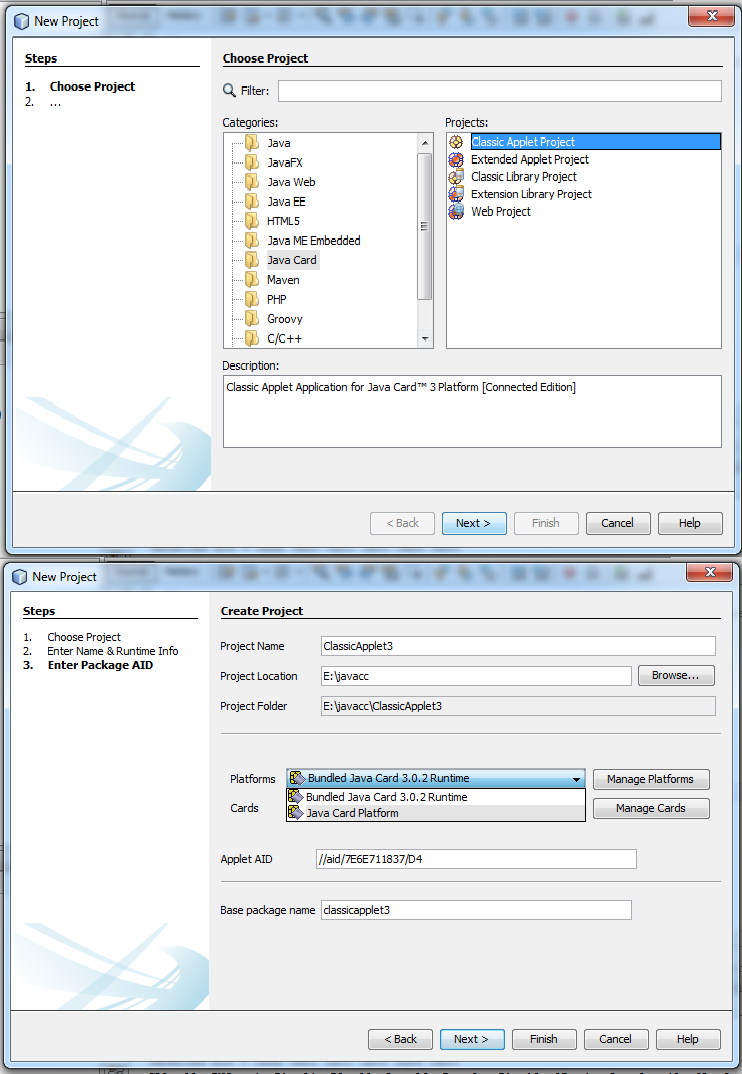
Install in Netbeans as plugin :- Tools -> Plugin Create an application

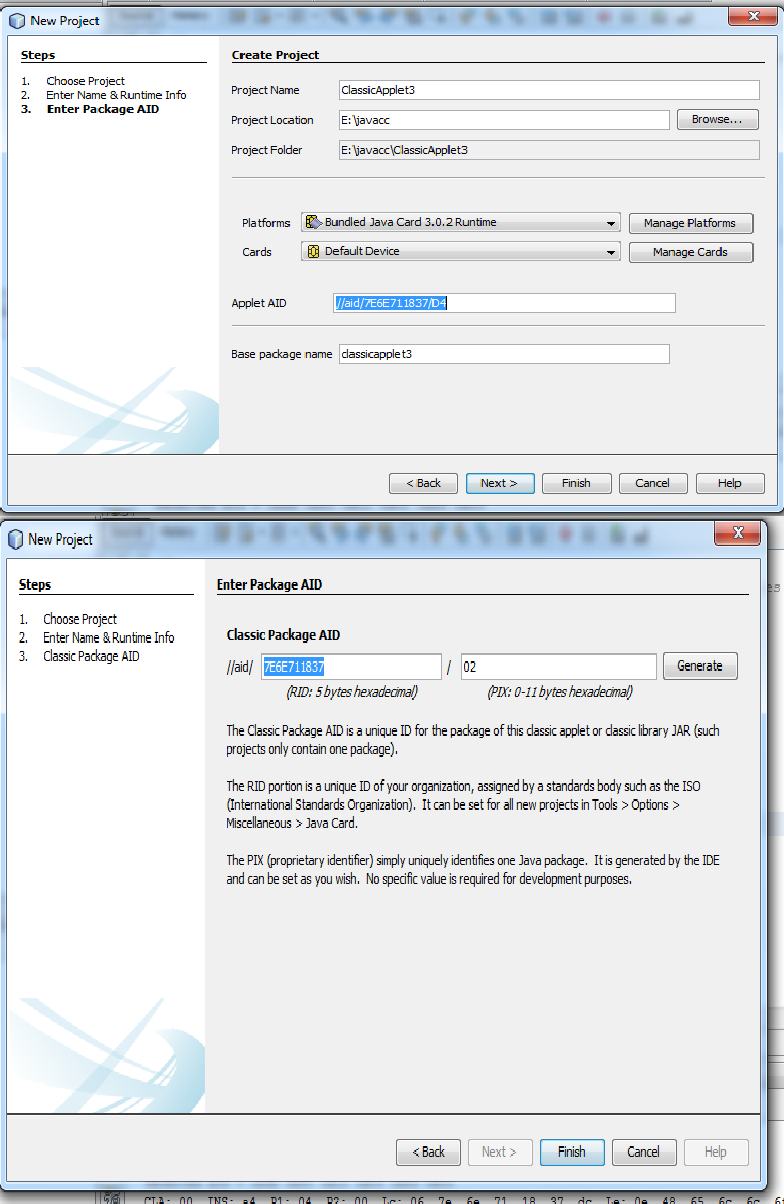
Install in Netbeans as

plugin :-Tools -> Plugin



**Create an application**





**Source Code:**

package classapp;

import javacard.framework.\*;

public class classapp extends Applet {

private byte[] received;

private static final short MAX\_LENGTH = 256;

private static final byte[] helloFidesmo =

{(byte)'H',(byte)'e',(byte)'l',(byte)'l',(byte)'o',(byte)'

',(byte)'F',(byte)'i',(byte)'d',(byte)'e',(byte)'s',(byte)'m',(

byte)'o',(byte)'!'};

public static void install(byte[] bArray, short bOffset, byte

bLength) {

new classapp();

}

protected classapp() {

received = new byte[MAX\_LENGTH];

register();

}

public void process(APDU apdu) {

//Insert your code here

byte buffer[] = apdu.getBuffer();

short length = (short) helloFidesmo.length;

Util.arrayCopyNonAtomic(helloFidesmo, (short)0, buffer,

(short)0,

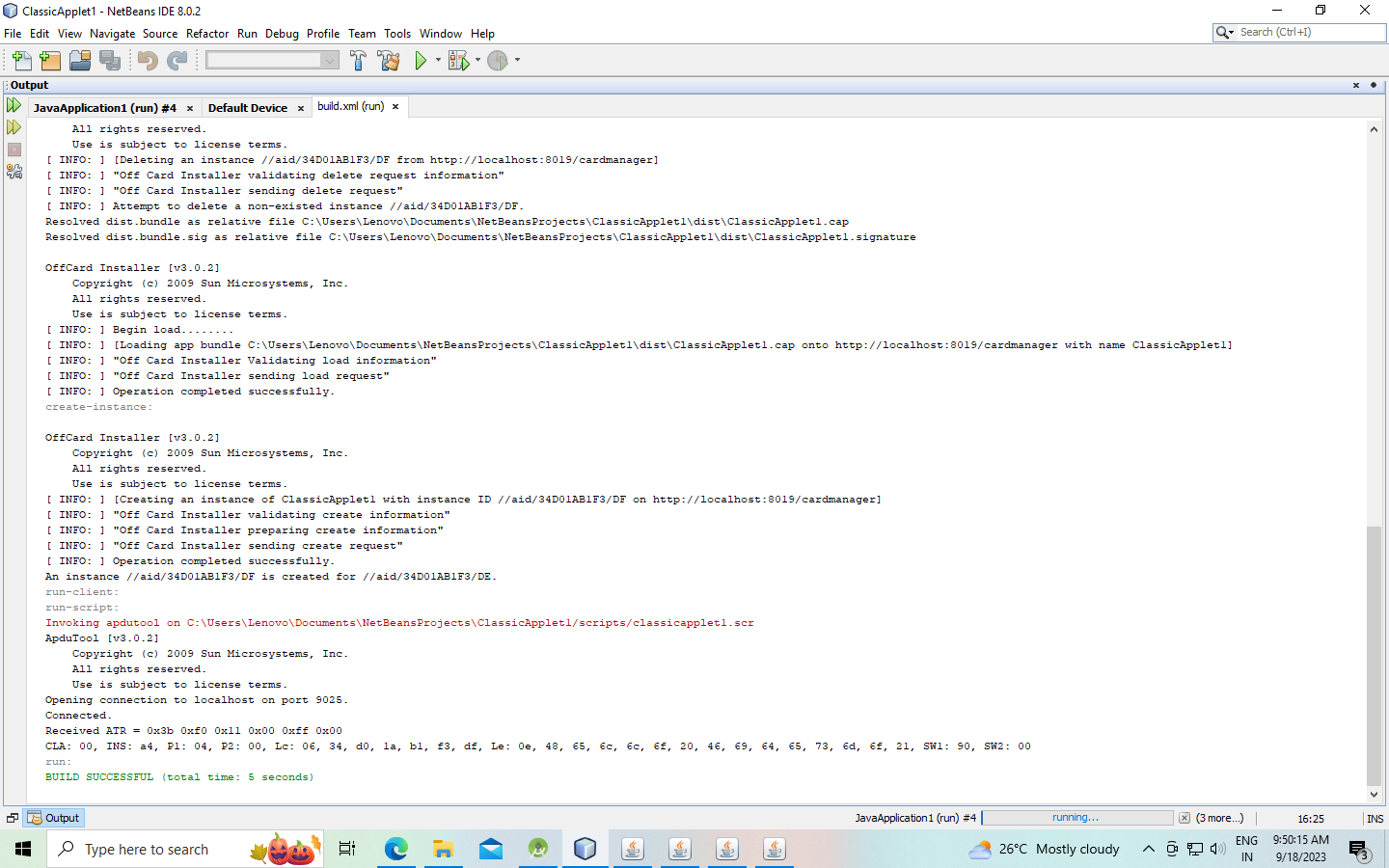
(short)length);

apdu.setOutgoingAndSend((short)0, length);

}31

}

**Output:**



**Conclusion:**Hence a java applet card was created and demonstrated successfully.