**MainActivity.java**

package com.jnu.lbsprivacy;  
import android.Manifest;  
import android.content.Context;  
import android.content.Intent;  
import android.content.SharedPreferences;  
import android.content.pm.PackageManager;  
import android.os.Build;  
import android.os.Bundle;  
import android.os.StrictMode;  
import android.util.Log;  
import android.view.MenuItem;  
import android.view.Menu;  
import android.widget.Toast;  
import com.google.android.material.navigation.NavigationView;  
import com.jnu.lbsprivacy.models.MyLocation;  
import com.jnu.lbsprivacy.algorithm.TraceMethod;  
import com.jnu.lbsprivacy.algorithm.GenerationBased;  
import com.jnu.lbsprivacy.algorithm.TraceRebuiltMethod;  
import com.jnu.lbsprivacy.ui.map.MapFragment;  
import com.jnu.lbsprivacy.utils.MultiprocessSharedPreferences;  
import com.jnu.lbsprivacy.utils.Path;  
import com.tbruyelle.rxpermissions3.BuildConfig;  
import com.tbruyelle.rxpermissions3.RxPermissions;  
import androidx.annotation.NonNull;  
import androidx.annotation.RequiresApi;  
import androidx.core.view.GravityCompat;  
import androidx.navigation.NavController;  
import androidx.navigation.Navigation;  
import androidx.navigation.fragment.NavHostFragment;  
import androidx.navigation.ui.AppBarConfiguration;  
import androidx.navigation.ui.NavigationUI;  
import androidx.drawerlayout.widget.DrawerLayout;  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.appcompat.widget.Toolbar;  
import androidx.preference.PreferenceManager;  
import java.io.File;  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.Collections;  
import java.util.List;  
public class MainActivity extends AppCompatActivity {  
 private AppBarConfiguration mAppBarConfiguration;  
 private Path loadedPath;  
 private static final String TAG = "MainActivity";  
 private MapFragment mapFragment;  
 private ArrayList<Path> trajectorySet;  
 GenerationBased generationBased = null;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();  
 StrictMode.setThreadPolicy(policy);  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.activity\_main);  
 Toolbar toolbar = findViewById(R.id.toolbar);  
 setSupportActionBar(toolbar);  
 DrawerLayout drawer = findViewById(R.id.drawer\_layout);  
 mAppBarConfiguration = new AppBarConfiguration.Builder(R.id.nav\_map)  
 .setDrawerLayout(drawer)  
 .build();  
 NavController navController = Navigation.findNavController(this, R.id.nav\_host\_fragment);  
 NavigationUI.setupActionBarWithNavController(this, navController, mAppBarConfiguration);  
 getPermissions();  
 PreferenceManager.setDefaultValues(this, R.xml.root\_preferences, false);  
 }  
 @Override  
 public void onStart() {  
 super.onStart();  
 NavHostFragment navHostFragment = (NavHostFragment) getSupportFragmentManager().findFragmentById(R.id.nav\_host\_fragment);  
 mapFragment = (MapFragment) navHostFragment.getChildFragmentManager().getFragments().get(0);  
 NavigationView navigationView = findViewById(R.id.nav\_view);  
 DrawerLayout drawer = findViewById(R.id.drawer\_layout);  
 navigationView.setNavigationItemSelectedListener(new NavigationView.OnNavigationItemSelectedListener() {  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 @Override  
 public boolean onNavigationItemSelected(@NonNull MenuItem item) {  
 long startTime, endTime;  
 double runTime;  
 String text;  
 switch (item.getItemId()) {  
 case R.id.nav\_algorithm3:  
 if (loadedPath == null) {  
 Toast.makeText(getApplicationContext(), "请先加载路径", Toast.LENGTH\_SHORT).show();  
 break;  
 }  
 startTime = System.currentTimeMillis();  
 TraceRebuiltMethod TraceRebuilt = new TraceRebuiltMethod(loadedPath);  
 Path rebuiltPath = TraceRebuilt.getResult();  
 endTime = System.currentTimeMillis();  
 runTime = (endTime - startTime) / 1000.0;  
 text = String.format("启发式隐私度量的隐私位置保护\n运行时长 %.3fs", runTime);  
 Toast.makeText(getApplicationContext(), text, Toast.LENGTH\_SHORT).show();  
 mapFragment.clearOverlay();  
 mapFragment.drawOrigPath(loadedPath);  
 mapFragment.drawNewPath(rebuiltPath);  
 break;  
 case R.id.nav\_algorithm1:  
 if (loadedPath == null) {  
 Toast.makeText(getApplicationContext(), "请先加载路径", Toast.LENGTH\_SHORT).show();  
 break;  
 }  
 startTime = System.currentTimeMillis();  
 TraceMethod traceMethod = new TraceMethod(loadedPath);  
 Path res3 = traceMethod.getResult();  
 endTime = System.currentTimeMillis();  
 runTime = (endTime - startTime) / 1000.0;  
 text = String.format("基于虚假位置的轨迹匿名\n运行时长 %.3fs", runTime);  
 Toast.makeText(getApplicationContext(), text, Toast.LENGTH\_SHORT).show();  
 mapFragment.clearOverlay();  
 mapFragment.drawOrigPath(loadedPath);  
 mapFragment.pathcorr(res3);  
 break;  
 case R.id.nav\_algorithm2:  
 if(trajectorySet == null)  
 Toast.makeText(getApplicationContext(), "请先点击多路径示例展示", Toast.LENGTH\_SHORT).show();  
 else {  
 startTime = System.currentTimeMillis();  
 Path paths = generationBased.getSingleResult();  
 Path initialPath = generationBased.getResultSingleOriginal();  
 Log.d("轨迹\_操作后", paths.toString());  
 Log.d("轨迹\_轨迹前", initialPath.toString());  
 endTime = System.currentTimeMillis();  
 runTime = (endTime - startTime) / 1000.0;  
 text = String.format("多路径匿名化重建算法\n运行时长 %.3fs", runTime);  
 Toast.makeText(getApplicationContext(), text, Toast.LENGTH\_SHORT).show();  
 mapFragment.clearOverlay();  
 if (initialPath.getWayPoints().size() >= 2) {  
 mapFragment.drawOrigPath(initialPath);  
 } else {  
 Log.d(TAG, "路径点不足");  
 }  
 if (paths.getWayPoints().size() >= 2) {  
 mapFragment.drawNewPath(paths);  
 } else {  
 Log.d(TAG, "路径点不足");  
 }  
 }  
 break;  
 case R.id.nav\_algorithm2\_1:  
 if(trajectorySet == null)  
 Toast.makeText(getApplicationContext(), "请先点击多路径示例展示", Toast.LENGTH\_SHORT).show();  
 else {  
 startTime = System.currentTimeMillis();  
 Path paths = generationBased.getSingleResultNotReconstruct();  
 Path initialPath2 = generationBased.getResultSingleOriginal();  
 endTime = System.currentTimeMillis();  
 runTime = (endTime - startTime) / 1000.0;  
 text = String.format("多路径匿名化算法\n运行时长 %.3fs", runTime);  
 Toast.makeText(getApplicationContext(), text, Toast.LENGTH\_SHORT).show();  
 mapFragment.clearOverlay();  
 if (initialPath2.getWayPoints().size() >= 2) {  
 mapFragment.drawOrigPath(initialPath2);  
 } else {  
 Log.d(TAG, "路径点不足");  
 }  
 if (paths.getWayPoints().size() >= 2) {  
 mapFragment.drawNewPath(paths);  
 } else {  
 Log.d(TAG, "路径点不足");  
 }  
 }  
 break;  
 case R.id.nav\_showroute:  
 if (loadedPath == null) {  
 Toast.makeText(getApplicationContext(), "请先加载路径", Toast.LENGTH\_SHORT).show();  
 break;  
 }  
 Toast.makeText(getApplicationContext(), "单路径示例展示", Toast.LENGTH\_SHORT).show();  
 mapFragment.clearOverlay();  
 mapFragment.drawOrigPath(loadedPath);  
 break;  
 case R.id.nav\_show\_multi\_route:  
 trajectorySet = randomLoadMultiGpxFile();  
 generationBased = new GenerationBased(trajectorySet);  
 Toast.makeText(getApplicationContext(), "多路径示例展示", Toast.LENGTH\_SHORT).show();  
 if (trajectorySet == null) {  
 Toast.makeText(getApplicationContext(), "请先加载路径", Toast.LENGTH\_SHORT).show();  
 break;  
 }  
 Path paths = generationBased.getResultSingleOriginal();  
 mapFragment.clearOverlay();  
 if (paths.getWayPoints().size() >= 2) {  
 mapFragment.drawOrigPath(paths);  
 }  
 break;  
 case R.id.nav\_clearroute:  
 Toast.makeText(getApplicationContext(), "路径清除", Toast.LENGTH\_SHORT).show();  
 mapFragment.clearOverlay();  
 break;  
 case R.id.nav\_upload\_route:  
 if (loadedPath != null) {  
 mapFragment.clearOverlay();  
 mapFragment.drawOrigPath(loadedPath);  
 loadedPath.upload();  
 }  
 else {  
 Toast.makeText(getApplicationContext(), "请先加载路径", Toast.LENGTH\_SHORT).show();  
 }  
 break;  
 case R.id.nav\_navigation:  
 Toast.makeText(getApplicationContext(), "路径生成", Toast.LENGTH\_SHORT).show();  
 mapFragment.setNavTrace();  
 break;  
 case R.id.fake\_location:  
 mapFragment.setInjectionModeOn();  
 break;  
 case R.id.position\_protect:  
 mapFragment.setLocationProtectOn();  
 break;  
 }  
 drawer.closeDrawer(GravityCompat.START);  
 return true;  
 }  
 });  
 }  
 @Override  
 public boolean onCreateOptionsMenu(Menu menu) {  
 getMenuInflater().inflate(R.menu.main\_activity2, menu);  
 return true;  
 }  
 @Override  
 public boolean onOptionsItemSelected(MenuItem item) {  
 Intent i;  
 switch (item.getItemId()) {  
 case R.id.action\_settings:  
 i = new Intent(getBaseContext(), SettingsActivity.class);  
 startActivity(i);  
 return true;  
 case R.id.action\_load\_gpx:  
 i = new Intent(getBaseContext(), LoadGPXActivity.class);  
 startActivityForResult(i, LoadGPXActivity.GET\_GPX\_FILENAME\_REQUEST);  
 return true;  
 default:  
 return super.onOptionsItemSelected(item);  
 }  
 }  
 @Override  
 public boolean onSupportNavigateUp() {  
 NavController navController = Navigation.findNavController(this, R.id.nav\_host\_fragment);  
 return NavigationUI.navigateUp(navController, mAppBarConfiguration)  
 || super.onSupportNavigateUp();  
 }  
 @Override  
 protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
 Bundle bundle = data.getExtras();  
 long startTime, endTime;  
 double runTime;  
 String text;  
 if (requestCode==LoadGPXActivity.GET\_GPX\_FILENAME\_REQUEST && resultCode == LoadGPXActivity.GET\_GPX\_FILENAME\_RESULT) {  
 String gpxFilename = bundle.getString(LoadGPXActivity.GPX\_FILENAME\_PARAM);  
 Log.d(TAG, "here: " + gpxFilename);  
 startTime = System.currentTimeMillis();  
 loadPathFromGPX(gpxFilename);  
 endTime = System.currentTimeMillis();  
 runTime = (endTime - startTime) / 1000.0;  
 text = String.format("加载路径耗时：%.3fs", runTime);  
 Toast.makeText(getApplicationContext(), text, Toast.LENGTH\_SHORT).show();  
 mapFragment.clearOverlay();  
 mapFragment.drawOrigPath(loadedPath);  
 }  
 super.onActivityResult(requestCode, resultCode, data);  
 }  
 private void getPermissions() {  
 RxPermissions rxPermissions = new RxPermissions(this);  
 rxPermissions.request(  
 Manifest.permission.ACCESS\_FINE\_LOCATION,  
 Manifest.permission.ACCESS\_COARSE\_LOCATION,  
 Manifest.permission.READ\_EXTERNAL\_STORAGE,  
 Manifest.permission.WRITE\_EXTERNAL\_STORAGE  
 ).subscribe(  
 granted -> {  
 if (granted) {  
 Toast.makeText(getApplicationContext(), "申请权限成功", Toast.LENGTH\_LONG).show();  
 }  
 else {  
 Toast.makeText(getApplicationContext(), "申请权限失败", Toast.LENGTH\_LONG).show();  
 }  
 }  
 );  
 }  
 private void loadPathFromGPX(String gpxFilename) {  
 loadedPath = new Path();  
 File externalDir = getApplicationContext().getExternalFilesDir("GPX");  
 gpxFilename = externalDir + "/" + gpxFilename;  
 File file = new File(gpxFilename);  
 try {  
 FileInputStream in = new FileInputStream(file);  
 loadedPath.loadFromGPX(in);  
 } catch (IOException e) {  
 Log.d("loadPathFromGPX", e.toString());  
 }  
 }  
 private ArrayList<Path> randomLoadMultiGpxFile() {  
 File externalDir = getApplicationContext().getExternalFilesDir("GPX");  
 List<File> fileList = new ArrayList(Arrays.asList(externalDir.listFiles((dir, name) -> name.contains("test"))));  
 Collections.shuffle(fileList);  
 ArrayList<Path> trajectorySet = new ArrayList<>();  
 String gpx\_names= "";  
 for (File file :fileList.subList(0, 4)) {  
 try {  
 gpx\_names += file.getName() + "\n";  
 Path trajectory = new Path();  
 FileInputStream in = new FileInputStream(file);  
 trajectory.loadFromGPX(in);  
 trajectorySet.add(trajectory);  
 in.close();  
 }  
 catch (IOException e) {  
 Log.e(TAG, e.toString());  
 }  
 }  
 return trajectorySet;  
 }  
 public void setLoadedPath(Path loadedPath) {  
 this.loadedPath = loadedPath;  
 }  
}

**Config.java：**

package com.jnu.lbsprivacy;  
public class Config {  
 public static final boolean RECT = false;  
}

**DemoApplication.java:**

package com.jnu.lbsprivacy;  
import android.app.Application;  
import com.baidu.mapapi.CoordType;  
import com.baidu.mapapi.SDKInitializer;  
public class DemoApplication extends Application {  
 @Override  
 public void onCreate() {  
 super.onCreate();  
 SDKInitializer.initialize(this);  
 SDKInitializer.setCoordType(CoordType.BD09LL);  
 }  
}

**LoadGPXActivity.java**

package com.jnu.lbsprivacy;  
import android.content.Intent;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.View;  
import android.widget.AdapterView;  
import android.widget.ArrayAdapter;  
import android.widget.ListView;  
import android.widget.Toast;  
import androidx.appcompat.app.ActionBar;  
import androidx.appcompat.app.AppCompatActivity;  
import java.io.File;  
import java.util.ArrayList;  
import java.util.Arrays;  
import org.apache.commons.io.comparator.NameFileComparator;  
public class LoadGPXActivity extends AppCompatActivity {  
 public static final int GET\_GPX\_FILENAME\_REQUEST = 0;  
 public static final int GET\_GPX\_FILENAME\_RESULT = 0;  
 public static final String GPX\_FILENAME\_PARAM = "GPX\_FILENAME\_PARAM";  
 private ListView mListView = null;  
 private ArrayList<String> gpxFileNames;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.load\_gpx\_activity);  
 gpxFileNames = new ArrayList<>();  
 mListView = (ListView)findViewById(R.id.gpx\_list\_view);  
 getExternalFilesList();  
 final ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,  
 android.R.layout.simple\_list\_item\_1, android.R.id.text1, gpxFileNames);  
 mListView.setAdapter(adapter);  
 mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
 @Override  
 public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
 String value = adapter.getItem(position);  
 Toast.makeText(getApplicationContext(), value + " 加载完成", Toast.LENGTH\_SHORT).show();  
 getSelectedGPXAndReturn(value);  
 }  
 });  
 ActionBar actionBar = getSupportActionBar();  
 if (actionBar != null) {  
 actionBar.setDisplayHomeAsUpEnabled(true);  
 }  
 }  
 private void getSelectedGPXAndReturn(String GPXFilename) {  
 Intent intent = getIntent();  
 intent.putExtra(GPX\_FILENAME\_PARAM, GPXFilename);  
 setResult(GET\_GPX\_FILENAME\_RESULT, intent);  
 this.finish();  
 }  
 private void getExternalFilesList() {  
 File externalDir = getApplicationContext().getExternalFilesDir("GPX");  
 Log.d("externalDir", externalDir.getPath());  
 File[] fileList = externalDir.listFiles((dir, name) -> name.toLowerCase().endsWith(".gpx"));  
 Arrays.sort(fileList, NameFileComparator.NAME\_COMPARATOR);  
 for (File file: fileList) {  
 gpxFileNames.add(file.getName());  
 }  
 }  
}

**SettingsActivity.java**

package com.jnu.lbsprivacy;  
import android.content.SharedPreferences;  
import android.os.Bundle;  
import android.view.View;  
import android.widget.Button;  
import android.widget.Toast;  
import androidx.appcompat.app.ActionBar;  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.preference.PreferenceFragmentCompat;  
import androidx.preference.PreferenceManager;  
import com.jnu.lbsprivacy.models.UserProfile;  
public class SettingsActivity extends AppCompatActivity {  
 public static final int SETTING\_REQUEST = 0;  
 private Button updateBtn;  
 UserProfile userProfile;  
 SharedPreferences sharedPref;  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 setContentView(R.layout.settings\_activity);  
 userProfile = new UserProfile();  
 userProfile.setUid("李德胜");  
 sharedPref = PreferenceManager.getDefaultSharedPreferences(this);  
 if (savedInstanceState == null) {  
 getSupportFragmentManager()  
 .beginTransaction()  
 .replace(R.id.settings, new SettingsFragment())  
 .commit();  
 }  
 ActionBar actionBar = getSupportActionBar();  
 if (actionBar != null) {  
 actionBar.setDisplayHomeAsUpEnabled(true);  
 }  
 updateBtn = findViewById(R.id.updateProfileBtn);  
 updateBtn.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 userProfile.setAge(Integer.valueOf(sharedPref.getString("age", "2")));  
 userProfile.setPurpose(Integer.valueOf(sharedPref.getString("purpose", "1")));  
 userProfile.setSensitivity(Integer.valueOf(sharedPref.getString("sensitivity", "1")));  
 userProfile.setLocation(Integer.valueOf(sharedPref.getString("location", "1")));  
 userProfile.setTrust(Integer.valueOf(sharedPref.getString("trust", "1")));  
 String protection\_level = userProfile.upload();  
 Toast.makeText(getApplicationContext(), "上传用户配置成功\n返回保护强度值: " + protection\_level, Toast.LENGTH\_SHORT).show();  
 }  
 });  
 }  
 public static class SettingsFragment extends PreferenceFragmentCompat {  
 @Override  
 public void onCreatePreferences(Bundle savedInstanceState, String rootKey) {  
 setPreferencesFromResource(R.xml.root\_preferences, rootKey);  
 }  
 }  
}

**XposedMod.java**

package com.jnu.lbsprivacy;  
import android.util.Log;  
import com.jnu.lbsprivacy.hooks.LMgetBestProviderHook;  
import com.jnu.lbsprivacy.hooks.LMgetLastKnownLocationHook;  
import com.jnu.lbsprivacy.hooks.LMgetLastLocationHook;  
import com.jnu.lbsprivacy.hooks.LMgetProvidersHook;  
import com.jnu.lbsprivacy.hooks.LMisProviderEnableHook;  
import com.jnu.lbsprivacy.hooks.LMremoveUpdate;  
import com.jnu.lbsprivacy.hooks.LMrequestLocationUpdateHook;  
import com.jnu.lbsprivacy.hooks.TMgetAllCellInfoHook;  
import com.jnu.lbsprivacy.hooks.TMgetCellLocationHook;  
import com.jnu.lbsprivacy.hooks.TMgetNeighboringCellInfoHook;  
import com.jnu.lbsprivacy.hooks.TMgetNetworkOperationHook;  
import com.jnu.lbsprivacy.hooks.TMgetPhoneTypeHook;  
import com.jnu.lbsprivacy.hooks.TMgetSimOperatorHook;  
import com.jnu.lbsprivacy.hooks.WMgetBSSIDHook;  
import com.jnu.lbsprivacy.hooks.WMgetScanResultHook;  
import java.util.List;  
import de.robv.android.xposed.IXposedHookLoadPackage;  
import de.robv.android.xposed.IXposedHookZygoteInit;  
import de.robv.android.xposed.callbacks.XC\_LoadPackage;  
public class XposedMod implements IXposedHookLoadPackage, IXposedHookZygoteInit {  
 private static final String TAG = XposedMod.class.getName();  
 public static final String prefilename = "FakeLocationPref";  
 private List<String> pkgNameList;  
 private void installWifiHooks(XC\_LoadPackage.LoadPackageParam loadPackageParam, String pkgName) {  
 new WMgetBSSIDHook(loadPackageParam.classLoader, pkgName).install();  
 new WMgetScanResultHook(loadPackageParam.classLoader, pkgName).install();  
 }  
 private void installCellHooks(XC\_LoadPackage.LoadPackageParam loadPackageParam, String pkgName) {  
 new TMgetPhoneTypeHook(loadPackageParam.classLoader, pkgName).install();  
 new TMgetCellLocationHook(loadPackageParam.classLoader, pkgName).install();  
 new TMgetNeighboringCellInfoHook(loadPackageParam.classLoader, pkgName).install();  
 new TMgetAllCellInfoHook(loadPackageParam.classLoader, pkgName).install();  
 new TMgetNetworkOperationHook(loadPackageParam.classLoader, pkgName).install();  
 new TMgetSimOperatorHook(loadPackageParam.classLoader, pkgName).install();  
 }  
 private void installLocationHooks(XC\_LoadPackage.LoadPackageParam loadPackageParam, String pkgName) {  
 new LMisProviderEnableHook(loadPackageParam.classLoader, pkgName).install();  
 new LMgetProvidersHook(loadPackageParam.classLoader, pkgName).install();  
 new LMgetBestProviderHook(loadPackageParam.classLoader, pkgName).install();  
 new LMrequestLocationUpdateHook(loadPackageParam.classLoader, pkgName).install();  
 new LMremoveUpdate(loadPackageParam.classLoader, pkgName).install();  
 new LMgetLastLocationHook(loadPackageParam.classLoader, pkgName).install();  
 new LMgetLastKnownLocationHook(loadPackageParam.classLoader, pkgName).install();  
 }  
 private void doAllHooks(XC\_LoadPackage.LoadPackageParam loadPackageParam, String pkgName) {  
 installWifiHooks(loadPackageParam, pkgName);  
 installCellHooks(loadPackageParam, pkgName);  
 installLocationHooks(loadPackageParam, pkgName);  
 }  
 @Override  
 public void handleLoadPackage(final XC\_LoadPackage.LoadPackageParam loadPackageParam) throws Throwable {  
 pkgNameList = List.of("com.dianping.v1", "me.ele", "com.autonavi.minimap", "com.sankuai.meituan");  
 if (pkgNameList.contains(loadPackageParam.packageName)) {  
 Log.d(TAG, "Find and Hook your pkg!!");  
 doAllHooks(loadPackageParam, loadPackageParam.packageName);  
 }  
 }  
 @Override  
 public void initZygote(IXposedHookZygoteInit.StartupParam startupParam) {  
 }  
}

**AnonymizationMethod.java**

package com.jnu.lbsprivacy.algorithm;  
import android.util.Log;  
import com.jnu.lbsprivacy.utils.Grid;  
import com.jnu.lbsprivacy.utils.Path;  
import java.time.ZonedDateTime;  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.Random;  
import io.jenetics.jpx.WayPoint;  
public class AnonymizationMethod {  
 final long REACHABLE\_DIS = 500;  
 final int CROSS\_CHANCE = 3;  
 private Path path;  
 private ArrayList<Grid> gridArrayList;  
 public AnonymizationMethod(Path p) {  
 path = p;  
 }  
 public Path getResult() {  
 Path res = new Path();  
 initNineGrids();  
 countWayPointInEachGrid();  
 res.addWayPoint(getBasePauseWayPoint());  
 Grid grid = new Grid(path);  
 while (res.getWayPoints().size() < path.getWayPoints().size()) {  
 WayPoint tmp = null;  
 do {  
 tmp = grid.getRandomWayPointInside();  
 } while (  
 !isWayPointReachable(  
 res.getWayPoints().get(res.getWayPoints().size() - 1),  
 tmp  
 )  
 );  
 res.addWayPoint(tmp);  
 WayPoint pShared = null;  
 Random rand = new Random();  
 for (int i=0; i < CROSS\_CHANCE; i++) {  
 pShared = path.getWayPoints().get(rand.nextInt(path.getWayPoints().size()));  
 if (isWayPointReachable(pShared, tmp)) {  
 res.addWayPoint(pShared);  
 Log.d("AnonymizationMethod", "Cross: " + pShared);  
 break;  
 }  
 }  
 }  
 return res;  
 }  
 private WayPoint getBasePauseWayPoint() {  
 Grid gBase = getGridWithMinCount();  
 ZonedDateTime tBase = path.getWayPoints().get(0).getTime().get();  
 WayPoint res = gBase.getRandomWayPointInside().toBuilder().time(tBase).build();  
 return res;  
 }  
 private void initNineGrids() {  
 gridArrayList = new ArrayList<Grid>();  
 Grid grid = new Grid(path);  
 double latDistance = grid.getLatMax() - grid.getLatMin();  
 double lngDistance = grid.getLngMax() - grid.getLngMin();  
 for (int lat\_step=0; lat\_step < 3; lat\_step ++) {  
 for (int lng\_step=0; lng\_step < 3; lng\_step++) {  
 Grid small\_grid = new Grid(  
 );  
 gridArrayList.add(small\_grid);  
 }  
 }  
 }  
 private void countWayPointInEachGrid() {  
 for (WayPoint wayPoint: path.getWayPoints()) {  
 for (Grid grid: gridArrayList) {  
 if (grid.isWayPointInside(wayPoint)) {  
 grid.setCount(grid.getCount() + 1);  
 }  
 }  
 }  
 }  
 private Grid getGridWithMinCount() {  
 return gridArrayList.stream()  
 .min(Comparator.comparing(Grid::getCount)).get();  
 }  
 private boolean isWayPointReachable(WayPoint dst, WayPoint src) {  
 return src.distance(dst).longValue() < REACHABLE\_DIS;  
 }  
}

**ExampleMethod.java**

package com.jnu.lbsprivacy.algorithm;  
import android.util.Log;  
import com.baidu.mapapi.model.LatLng;  
import com.baidu.mapapi.search.core.PoiInfo;  
import com.baidu.mapapi.search.core.SearchResult;  
import com.baidu.mapapi.search.poi.OnGetPoiSearchResultListener;  
import com.baidu.mapapi.search.poi.PoiDetailResult;  
import com.baidu.mapapi.search.poi.PoiDetailSearchResult;  
import com.baidu.mapapi.search.poi.PoiIndoorInfo;  
import com.baidu.mapapi.search.poi.PoiIndoorResult;  
import com.baidu.mapapi.search.poi.PoiNearbySearchOption;  
import com.baidu.mapapi.search.poi.PoiResult;  
import com.baidu.mapapi.search.poi.PoiSearch;  
import com.jnu.lbsprivacy.utils.Path;  
import java.util.ArrayList;  
import java.util.Collections;  
import io.jenetics.jpx.WayPoint;  
public class ExampleMethod {  
 private Path path;  
 public ExampleMethod(Path p) {  
 path = p;  
 }  
 public Path getResult() {  
 ArrayList<WayPoint> orig = (ArrayList<WayPoint>) path.getWayPoints().clone();  
 Collections.shuffle(orig);  
 return new Path(orig);  
 }  
 private class MYPOISearchResult implements OnGetPoiSearchResultListener {  
 @Override  
 public void onGetPoiResult(PoiResult poiResult) {  
 if (poiResult == null || poiResult.error == SearchResult.ERRORNO.RESULT\_NOT\_FOUND) {  
 Log.d("POIResult", "No Result");  
 return;  
 }  
 for (PoiInfo poiInfo: poiResult.getAllPoi()) {  
 Log.d("POIResult", poiInfo.name + ", " + poiInfo.location.toString());  
 }  
 }  
 @Override  
 public void onGetPoiDetailResult(PoiDetailResult poiDetailResult) {  
 Log.d("POIResult", poiDetailResult.toString());  
 }  
 @Override  
 public void onGetPoiDetailResult(PoiDetailSearchResult poiDetailSearchResult) {  
 Log.d("POIResult", poiDetailSearchResult.toString());  
 }  
 @Override  
 public void onGetPoiIndoorResult(PoiIndoorResult poiIndoorResult) {  
 }  
 }  
 public void testPOI(LatLng center) {  
 PoiSearch poiSearch = PoiSearch.newInstance();  
 poiSearch.setOnGetPoiSearchResultListener(new MYPOISearchResult());  
 PoiNearbySearchOption options = new PoiNearbySearchOption().keyword("酒店$餐厅").location(center).radius(1000).pageNum(0);  
 poiSearch.searchNearby(options);  
 }  
}

**GenerationBased.java**

package com.jnu.lbsprivacy.algorithm;  
import android.os.Build;  
import android.util.Log;  
import androidx.annotation.RequiresApi;  
import com.baidu.mapapi.model.LatLng;  
import com.baidu.mapapi.utils.DistanceUtil;  
import com.jnu.lbsprivacy.utils.DouglasPeucker;  
import com.jnu.lbsprivacy.utils.Path;  
import com.jnu.lbsprivacy.utils.TraceCorrection;  
import java.time.ZonedDateTime;  
import java.util.ArrayList;  
import java.util.Collection;  
import java.util.Iterator;  
import java.util.List;  
import java.util.Optional;  
import java.util.Random;  
import io.jenetics.jpx.WayPoint;  
class Bound<T> {  
 private double min;  
 private double max;  
 Bound(double \_min, double \_max) {  
 min = \_min;  
 max = \_max;  
 }  
 double getMin() {  
 return min;  
 }  
 double getMax() {  
 return max;  
 }  
 void setMin(double \_min) {  
 min = \_min;  
 }  
 void setMax(double \_max) {  
 max = \_max;  
 }  
}  
class BoundPoint{  
 private Bound boundLat;  
 private Bound boundLng;  
 private Bound boundTime;  
 BoundPoint() {  
 }  
 void setBoundLat(Bound lat) {  
 boundLat = lat;  
 }  
 void setBoundLng(Bound lng) {  
 boundLng = lng;  
 }  
 void setBoundTime(Bound time) {  
 boundTime = time;  
 }  
 Bound getLat() {  
 return boundLat;  
 }  
 Bound getLng() {  
 return boundLng;  
 }  
 Bound getTime() {  
 return boundTime;  
 }  
}  
class BoundTrajectory extends ArrayList<BoundPoint>{  
 BoundTrajectory() {  
 }  
 BoundTrajectory(Collection<? extends BoundPoint> c){  
 }  
}  
class BoundTrajectorySet extends ArrayList<BoundTrajectory>{  
 BoundTrajectorySet() {  
 }  
 BoundTrajectorySet(Collection<? extends BoundTrajectory> c){  
 }  
}  
public class GenerationBased {  
 private BoundTrajectorySet notReconstructResult = new BoundTrajectorySet();  
 private BoundTrajectory trajectoryRecord1 = new BoundTrajectory();  
 private BoundTrajectory trajectoryRecord2 = new BoundTrajectory();  
 private ArrayList<WayPoint> startPointList = new ArrayList<>();  
 private ArrayList<WayPoint> endPointList = new ArrayList<>();  
 private ArrayList<Path> origTrajectorySet = null;  
 private BoundTrajectorySet boundTrajectorySet = new BoundTrajectorySet();  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 public GenerationBased(ArrayList<Path> trajectorySet) {  
 origTrajectorySet = trajectorySet;  
 ArrayList<Path> reducedTrajectorySet= DouglasPeucker.runMulti(trajectorySet, 1e-3d);  
 startPointList.clear();  
 endPointList.clear();  
 for (Path trajectory: reducedTrajectorySet) {  
 WayPoint startPoint = trajectory.getWayPoints().get(0);  
 WayPoint endPoint = trajectory.getWayPoints().get(trajectory.getWayPoints().size() - 1);  
 startPointList.add(startPoint);  
 endPointList.add(endPoint);  
 ArrayList<WayPoint> wayPoint = trajectory.getWayPoints();  
 BoundTrajectory boundTrajectory = new BoundTrajectory();  
 for (WayPoint point: wayPoint) {  
 double lat = point.getLatitude().doubleValue();  
 double lng = point.getLongitude().doubleValue();  
 double timestamp = 0;  
 Optional<ZonedDateTime> timeLog = point.getTime();  
 timestamp = (double) timeLog.get().toInstant().getEpochSecond();  
 BoundPoint bp = new BoundPoint();  
 bp.setBoundLat(new Bound(lat - 0.0001, lat + 0.0001));  
 bp.setBoundLng(new Bound(lng - 0.0001, lng + 0.0001));  
 bp.setBoundTime(new Bound(timestamp - 5 <= 0 ? 0 : timestamp-5, timestamp + 5));  
 boundTrajectory.add(bp);  
 }  
 boundTrajectorySet.add(boundTrajectory);  
 }  
 }  
 private int randomlyChoose(int num){  
 return new Random().nextInt(num);  
 }  
 private BoundTrajectorySet multiTGA(BoundTrajectorySet origSet, int k, boolean isFast, boolean isReconstruction){  
 BoundTrajectorySet result = new BoundTrajectorySet();  
 BoundTrajectorySet G = new BoundTrajectorySet();  
 BoundTrajectory repG = new BoundTrajectory();  
 do{  
 G.clear();  
 repG.clear();  
 int randFirst = randomlyChoose(origSet.size());  
 BoundTrajectory tr = origSet.get(randFirst);  
 G.add(tr);  
 repG = tr;  
 do{  
 BoundTrajectorySet TRSubG = subtraction(origSet,G);  
 BoundTrajectory trPrime = findTheClosestTrajectory(TRSubG, repG);  
 G.add(trPrime);  
 if (!isFast){  
 BoundTrajectorySet tempSet = new BoundTrajectorySet();  
 tempSet.add(repG);  
 tempSet.add(trPrime);  
 repG = anonTrajectory(tempSet).get(0);  
 }  
 }while(G.size()<k);  
 BoundTrajectorySet newResult = anonTrajectory(G);  
 notReconstructResult.addAll(newResult);  
 if (isReconstruction){  
 newResult = reconstruction(newResult, k);  
 }  
 result.addAll(newResult);  
 origSet = subtraction(origSet,G);  
 }while(origSet.size()>=k);  
 return result;  
 }  
 private BoundTrajectorySet multiTGADisplaySingle(BoundTrajectorySet origSet, int k, boolean isFast, boolean isReconstruction){  
 BoundTrajectorySet result = new BoundTrajectorySet();  
 BoundTrajectorySet G = new BoundTrajectorySet();  
 BoundTrajectory repG = new BoundTrajectory();  
 G.clear();  
 repG.clear();  
 BoundTrajectory tr = origSet.get(randFirst);  
 G.add(tr);  
 repG = tr;  
 do{  
 BoundTrajectorySet TRSubG = subtraction(origSet,G);  
 BoundTrajectory trPrime = findTheClosestTrajectory(TRSubG, repG);  
 G.add(trPrime);  
 if (!isFast){  
 BoundTrajectorySet tempSet = new BoundTrajectorySet();  
 tempSet.add(repG);  
 tempSet.add(trPrime);  
 repG = anonTrajectory(tempSet).get(0);  
 }  
 }while(G.size()<k);  
 BoundTrajectorySet newResult = anonTrajectory(G);  
 notReconstructResult.addAll(newResult);  
 if (isReconstruction){  
 newResult = reconstruction(newResult, k);  
 }  
 result.addAll(newResult);  
 return result;  
 }  
 private BoundTrajectorySet reconstruction(BoundTrajectorySet shouldBeReconstruct, int k) {  
 BoundTrajectorySet reconstructionSet = new BoundTrajectorySet();  
 BoundTrajectory trajectory = shouldBeReconstruct.get(0);  
 Iterator<BoundPoint> trajectoryIterator = trajectory.iterator();  
 double averageLat = 0;  
 double averageLng = 0;  
 int times = 0;  
 double preAverageLat;  
 double preAverageLng;  
 while(trajectoryIterator.hasNext()){  
 BoundPoint nextTrajectory = trajectoryIterator.next();  
 preAverageLat = (nextTrajectory.getLat().getMin() + nextTrajectory.getLat().getMax()) / 2;  
 preAverageLng = (nextTrajectory.getLng().getMin() + nextTrajectory.getLng().getMax()) / 2;  
 if(trajectoryIterator.hasNext()) {  
 BoundPoint next2Trajectory = trajectoryIterator.next();  
 averageLat += Math.abs((next2Trajectory.getLat().getMin() + next2Trajectory.getLat().getMax()) / 2 - preAverageLat);  
 averageLng += Math.abs((next2Trajectory.getLng().getMin() + next2Trajectory.getLng().getMax()) / 2 - preAverageLng);  
 times++;  
 }  
 }  
 averageLat = averageLat/times;  
 averageLng = averageLng/times;  
 for (int i = 0; i < k; i++) {  
 BoundTrajectory reconstructionTrajectory = new BoundTrajectory();  
 for (BoundPoint trajectoryPoint : trajectory) {  
 BoundPoint bp = new BoundPoint();  
 double latRandom = doubleRandom((trajectoryPoint.getLat().getMin() + trajectoryPoint.getLat().getMax()) / 2 - averageLat,  
 (trajectoryPoint.getLat().getMax() + trajectoryPoint.getLat().getMin()) / 2 + averageLat);  
 Bound lat = new Bound(latRandom - 0.0001, latRandom + 0.0001);  
 double lngRandom = doubleRandom((trajectoryPoint.getLng().getMin() + trajectoryPoint.getLng().getMax()) / 2 - averageLng,  
 (trajectoryPoint.getLng().getMin() + trajectoryPoint.getLng().getMax()) / 2 + averageLng);  
 Bound lng = new Bound(lngRandom - 0.0001, lngRandom + 0.0001);  
 double timeRandom = doubleRandom(trajectoryPoint.getTime().getMin(), trajectoryPoint.getTime().getMax());  
 Bound time = new Bound(timeRandom - 5 <=0 ? 0 : timeRandom - 5, timeRandom + 5);  
 Log.d("测试重构lat", String.valueOf(latRandom));  
 Log.d("测试重构lng", String.valueOf(lngRandom));  
 bp.setBoundLat(lat);  
 bp.setBoundLng(lng);  
 bp.setBoundTime(time);  
 reconstructionTrajectory.add(bp);  
 }  
 reconstructionSet.add(reconstructionTrajectory);  
 }  
 return reconstructionSet;  
 }  
 private double doubleRandom(double min, double max) {  
 }  
 private BoundTrajectory findTheClosestTrajectory(BoundTrajectorySet set, BoundTrajectory traj){  
 double minValue = Double.POSITIVE\_INFINITY;  
 BoundTrajectory trPrime = new BoundTrajectory();  
 for (BoundTrajectory each: set){  
 double DSTResult = DSTorOPT(each,traj);  
 if(DSTResult < minValue){  
 minValue = DSTResult;  
 trPrime = each;  
 }  
 }  
 return trPrime;  
 }  
 private BoundTrajectorySet subtraction(BoundTrajectorySet set1, BoundTrajectorySet set2){  
 BoundTrajectorySet setTemp = new BoundTrajectorySet(set1);  
 Iterator<BoundTrajectory> it = setTemp.iterator();  
 while(it.hasNext()) {  
 BoundTrajectory curr = it.next();  
 if(set2.contains(curr)) {  
 it.remove();  
 }  
 }  
 return setTemp;  
 }  
 private BoundTrajectorySet suppressUnmatched(BoundTrajectorySet M, BoundTrajectory trPrime){  
 BoundTrajectorySet MTemp = new BoundTrajectorySet(M);  
 ArrayList<Integer> shouldBeDelete = new ArrayList<>();  
 for(BoundPoint primePoint : trPrime){  
 if(!trajectoryRecord1.contains(primePoint)){  
 int index = trPrime.indexOf(primePoint);  
 shouldBeDelete.add(index);  
 }  
 }  
 for(int i = shouldBeDelete.size() -1; i>=0;i--){  
 for (BoundTrajectory eachMTemp : MTemp) {  
 eachMTemp.remove((int)shouldBeDelete.get(i));  
 }  
 }  
 return MTemp;  
 }  
 private BoundTrajectory removeNotExist(BoundTrajectory tra1){  
 BoundTrajectory traTemp = new BoundTrajectory(tra1);  
 Iterator<BoundPoint> it = traTemp.iterator();  
 while(it.hasNext()) {  
 BoundPoint curr = it.next();  
 if(!trajectoryRecord2.contains(curr)) {  
 it.remove();  
 }  
 }  
 return traTemp;  
 }  
 private double sigmaLCM(BoundPoint p1, BoundPoint p2){  
 double result;  
 double S;  
 double T;  
 double U;  
 if (p2 == null){  
 double diagonal1 = DistanceUtil.getDistance(  
 new LatLng(p1.getLat().getMin(),p1.getLng().getMin()),  
 new LatLng(p1.getLat().getMax(),p1.getLng().getMax()));  
 double diagonal2 = DistanceUtil.getDistance(  
 new LatLng(p1.getLat().getMin(),p1.getLng().getMax()),  
 new LatLng(p1.getLat().getMax(),p1.getLng().getMin()));  
 double diagonal = Math.max(diagonal1, diagonal2);  
 result = Math.log(U);  
 }else{  
 double maxLat = Math.max(p1.getLat().getMax(),p2.getLat().getMax());  
 double minLat = Math.min(p1.getLat().getMin(),p2.getLat().getMin());  
 double maxLng = Math.max(p1.getLng().getMax(),p2.getLng().getMax());  
 double minLng = Math.min(p1.getLng().getMin(),p2.getLng().getMin());  
 double length = DistanceUtil.getDistance(new LatLng(minLat,minLng), new LatLng(maxLat,minLng));  
 double width = DistanceUtil.getDistance(new LatLng(minLat,minLng), new LatLng(minLat,maxLng));  
 T = Math.max(Math.max(p1.getTime().getMax(),p2.getTime().getMax()), Math.max(p1.getTime().getMin(),p2.getTime().getMin())) -  
 Math.min(Math.min(p1.getTime().getMax(),p2.getTime().getMax()), Math.min(p1.getTime().getMin(),p2.getTime().getMin()));  
 result = Math.log(U);  
 }  
 return result;  
 }  
 private double DSTorOPT(BoundTrajectory tr1, BoundTrajectory tr2){  
 int rowSize = tr1.size() + 1;  
 int colSize = tr2.size() + 1;  
 double[][] dpArray = new double[rowSize][colSize];  
 dpArray[0][0] = 0;  
 for(int j = 1; j < colSize; j++){  
 dpArray[0][j] = 0;  
 for (int tr2PointTime = colSize - 2; tr2PointTime >= colSize - j - 1; tr2PointTime--){  
 dpArray[0][j] += sigmaLCM(tr2.get(tr2PointTime),null);  
 }  
 }  
 for(int i = 1; i < rowSize; i++){  
 dpArray[i][0] = 0;  
 for (int tr1PointTime = rowSize - 2; tr1PointTime >= rowSize - i - 1; tr1PointTime--){  
 dpArray[i][0] += sigmaLCM(tr1.get(tr1PointTime),null);  
 }  
 }  
 for(int i = 1; i < rowSize; i++){  
 for(int j = 1; j < colSize; j++){  
 double value1 = dpArray[i-1][j-1] + sigmaLCM(tr1.get(rowSize - i - 1),tr2.get(colSize - j - 1));  
 double value2 = dpArray[i][j-1] + sigmaLCM(tr2.get(colSize - j - 1),null);  
 double value3 = dpArray[i-1][j] + sigmaLCM(tr1.get(rowSize - i - 1),null);  
 if (value1 < minValue23){  
 if (!trajectoryRecord1.contains(tr1.get(rowSize - i - 1)) && !trajectoryRecord2.contains(tr2.get(colSize - j - 1))){  
 trajectoryRecord1.add(tr1.get(rowSize - i - 1));  
 trajectoryRecord2.add(tr2.get(colSize - j - 1));  
 }  
 dpArray[i][j] = value1;  
 }else{  
 dpArray[i][j] = minValue23;  
 }  
 }  
 }  
 return dpArray[rowSize-1][colSize-1];  
 }  
 private BoundTrajectory searchAnon(BoundTrajectorySet M){  
 BoundTrajectory trPrime = new BoundTrajectory();  
 if (M.size()==1){  
 trPrime = M.get(0);  
 }else{  
 for(int i = 0;i < Math.min(M.get(0).size(),M.get(1).size()); i++){  
 double maxValueLat = Double.NEGATIVE\_INFINITY;  
 double minValueLat = Double.POSITIVE\_INFINITY;  
 double maxValueLng = Double.NEGATIVE\_INFINITY;  
 double minValueLng = Double.POSITIVE\_INFINITY;  
 double maxValueTime = Double.NEGATIVE\_INFINITY;  
 double minValueTime = Double.POSITIVE\_INFINITY;  
 BoundPoint point = new BoundPoint();  
 for(BoundTrajectory trajectory : M){  
 if(trajectory.get(i).getLat().getMax() > maxValueLat){  
 maxValueLat = trajectory.get(i).getLat().getMax();  
 }  
 if(trajectory.get(i).getLat().getMin() < minValueLat){  
 minValueLat = trajectory.get(i).getLat().getMin();  
 }  
 if(trajectory.get(i).getLng().getMax() > maxValueLng){  
 maxValueLng = trajectory.get(i).getLng().getMax();  
 }  
 if(trajectory.get(i).getLng().getMin() < minValueLng){  
 minValueLng = trajectory.get(i).getLng().getMin();  
 }  
 if(trajectory.get(i).getTime().getMax() > maxValueTime){  
 maxValueTime = trajectory.get(i).getTime().getMax();  
 }  
 if(trajectory.get(i).getTime().getMin() < minValueTime){  
 minValueTime = trajectory.get(i).getTime().getMin();  
 }  
 }  
 Bound Lat = new Bound(minValueLat, maxValueLat);  
 Bound Lng = new Bound(minValueLng, maxValueLng);  
 Bound Time = new Bound(minValueTime, maxValueTime);  
 point.setBoundLat(Lat);  
 point.setBoundLng(Lng);  
 point.setBoundTime(Time);  
 trPrime.add(point);  
 }  
 }  
 return trPrime;  
 }  
 private BoundTrajectorySet anonTrajectory(BoundTrajectorySet G){  
 BoundTrajectory trM = calculatePairwise(G);  
 BoundTrajectorySet M = new BoundTrajectorySet();  
 M.add(trM);  
 do{  
 BoundTrajectory trPrime = searchAnon(M);  
 BoundTrajectorySet GSubM = subtraction(G,M);  
 BoundTrajectory trNew = GSubM.get(randomlyChoose(GSubM.size()));  
 trajectoryRecord1.clear();  
 trajectoryRecord2.clear();  
 DSTorOPT(trPrime,trNew);  
 M = suppressUnmatched(M, trPrime);  
 M.add(trNew);  
 }while(M.size()<G.size());  
 M = replacePoints(M);  
 return M;  
 }  
 private BoundTrajectorySet replacePoints(BoundTrajectorySet M){  
 BoundTrajectorySet MTemp = new BoundTrajectorySet(M);  
 int size = Math.min(MTemp.get(0).size(),MTemp.get(1).size());  
 for(int i = 0; i < size; i++){  
 double maxValueLat = Double.NEGATIVE\_INFINITY;  
 double minValueLat = Double.POSITIVE\_INFINITY;  
 double maxValueLng = Double.NEGATIVE\_INFINITY;  
 double minValueLng = Double.POSITIVE\_INFINITY;  
 double maxValueTime = Double.NEGATIVE\_INFINITY;  
 double minValueTime = Double.POSITIVE\_INFINITY;  
 BoundPoint point = new BoundPoint();  
 for(BoundTrajectory trajectory : MTemp){  
 if(trajectory.get(i).getLat().getMax() > maxValueLat){  
 maxValueLat = trajectory.get(i).getLat().getMax();  
 }  
 if(trajectory.get(i).getLat().getMin() < minValueLat){  
 minValueLat = trajectory.get(i).getLat().getMin();  
 }  
 if(trajectory.get(i).getLng().getMax() > maxValueLng){  
 maxValueLng = trajectory.get(i).getLng().getMax();  
 }  
 if(trajectory.get(i).getLng().getMin() < minValueLng){  
 minValueLng = trajectory.get(i).getLng().getMin();  
 }  
 if(trajectory.get(i).getTime().getMax() > maxValueTime){  
 maxValueTime = trajectory.get(i).getTime().getMax();  
 }  
 if(trajectory.get(i).getTime().getMin() < minValueTime){  
 minValueTime = trajectory.get(i).getTime().getMin();  
 }  
 }  
 Bound Lat = new Bound(minValueLat, maxValueLat);  
 Bound Lng = new Bound(minValueLng, maxValueLng);  
 Bound Time = new Bound(minValueTime, maxValueTime);  
 point.setBoundLat(Lat);  
 point.setBoundLng(Lng);  
 point.setBoundTime(Time);  
 for(BoundTrajectory trajectory : MTemp) {  
 trajectory.set(i, point);  
 }  
 }  
 return MTemp;  
 }  
 private BoundTrajectory calculatePairwise(BoundTrajectorySet G){  
 List<Double> totalNumSet = new ArrayList<Double>();  
 for(int i=0;i<G.size();i++){  
 double totalNum = 0;  
 for(int j=0;j<G.size();j++){  
 if(i==j){  
 continue;  
 }  
 totalNum += DSTorOPT(G.get(i),G.get(j));  
 }  
 totalNumSet.add(totalNum);  
 }  
 int minIndex = 0;  
 double minValue = Double.POSITIVE\_INFINITY;  
 for (int i=0;i<totalNumSet.size();i++){  
 if(totalNumSet.get(i) < minValue){  
 minIndex = i;  
 minValue = totalNumSet.get(i);  
 }  
 }  
 return G.get(minIndex);  
 }  
 private static WayPoint LatLngCalculateMidPoint(double lat1, double lng1, double lat2, double lng2) {  
 double dLng = Math.toRadians(lng2 - lng1);  
 lat1 = Math.toRadians(lat1);  
 lat2 = Math.toRadians(lat2);  
 lng1 = Math.toRadians(lng1);  
 double lng3 = lng1 + Math.atan2(By, Math.cos(lat1) + Bx);  
 WayPoint wp = WayPoint.builder().lat(Math.toDegrees(lat3)).lon(Math.toDegrees(lng3)).build();  
 return wp;  
 }  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 private ArrayList<Path> transToPath(BoundTrajectorySet trajectorySet){  
 ArrayList<Path> wayPointsList = new ArrayList<>();  
 int index = 0;  
 for (BoundTrajectory trajectory : trajectorySet){  
 ArrayList<WayPoint> wayPoints = new ArrayList<>();  
 wayPoints.add(startPointList.get(index));  
 for(BoundPoint point : trajectory){  
 WayPoint pointNeeded = LatLngCalculateMidPoint(point.getLat().getMin(),point.getLng().getMin(),  
 point.getLat().getMax(), point.getLng().getMax());  
 wayPoints.add(pointNeeded);  
 }  
 wayPoints.add(endPointList.get(index));  
 index++;  
 TraceCorrection correction = new TraceCorrection(new Path(wayPoints));  
 wayPointsList.add(correction.getresult());  
 wayPointsList.add(new Path(wayPoints));  
 }  
 return wayPointsList;  
 }  
 public void clearCache(){  
 notReconstructResult.clear();  
 trajectoryRecord1.clear();  
 trajectoryRecord2.clear();  
 }  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 public ArrayList<Path> getResult() {  
 clearCache();  
 int k = 2;  
 BoundTrajectorySet multiTrajectory = multiTGA(boundTrajectorySet, k,false, true);  
 return transToPath(multiTrajectory);  
 }  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 public Path getSingleResult(){  
 clearCache();  
 int k = 2;  
 BoundTrajectorySet multiTrajectory = multiTGADisplaySingle(boundTrajectorySet, k,false, true);  
 return transToPath(multiTrajectory).get(0);  
 }  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 public Path getSingleResultNotReconstruct(){  
 clearCache();  
 int k = 2;  
 multiTGADisplaySingle(boundTrajectorySet, k,false, false);  
 return transToPath(notReconstructResult).get(0);  
 }  
 public Path getResultSingleOriginal() {  
 return origTrajectorySet.get(0);  
 }  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 public ArrayList<Path> getResultNotReconstruct(){  
 clearCache();  
 int k = 2;  
 multiTGA(boundTrajectorySet, k,false, false);  
 return transToPath(notReconstructResult);  
 }  
 public ArrayList<Path> getResultOriginal() {  
 return origTrajectorySet;  
 }  
}

**LocationProtectionMethod.java**

package com.jnu.lbsprivacy.algorithm;  
import android.util.Log;  
import com.baidu.mapapi.model.LatLng;  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.lang.reflect.Array;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.util.ArrayList;  
import java.util.Random;  
public class LocationProtectionMethod {  
 private LatLng input;  
 private static final int RANDOM\_POINT\_NUM = 3;  
 private static final double randomParam = 0.005;  
 public LocationProtectionMethod(LatLng point) {  
 input = point;  
 }  
 private static ArrayList<LatLng> randomlyChoosePoiByCircle(double lat1, double lng1, int radius){  
 HttpURLConnection connection = null;  
 BufferedReader reader = null;  
 try {  
 URL url = new URL(stringQuery);  
 connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
 connection.setConnectTimeout(50000);  
 connection.setReadTimeout(50000);  
 InputStream in = connection.getInputStream();  
 reader = new BufferedReader(new InputStreamReader(in));  
 StringBuilder result = new StringBuilder();  
 String line;  
 while ((line = reader.readLine()) != null) {  
 result.append(line);  
 }  
 JSONObject jsonObject = new JSONObject(String.valueOf(result));  
 JSONArray jsonArrayResults = jsonObject.getJSONArray("results");  
 int lenResults = jsonArrayResults.length();  
 ArrayList<LatLng> retPoints = new ArrayList<>();  
 if (lenResults == 0){  
 return null;  
 }else{  
 for(int i = 0; i< lenResults;i++){  
 JSONObject randValue = jsonArrayResults.getJSONObject(i);  
 JSONObject location = randValue.getJSONObject("location");  
 double lat = location.getDouble("lat");  
 double lng = location.getDouble("lng");  
 retPoints.add(new LatLng(lat,lng));  
 }  
 }  
 return retPoints;  
 } catch (IOException | JSONException e) {  
 Log.d("eValue", e.toString());  
 e.printStackTrace();  
 } finally {  
 if (reader != null) {  
 try {  
 reader.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 connection.disconnect();  
 }  
 }  
 return null;  
 }  
 public ArrayList<LatLng> getResult(){  
 ArrayList<LatLng> poiResult = randomlyChoosePoiByCircle(input.latitude, input.longitude, 500);  
 if (poiResult == null) {  
 for (int i=0; i < RANDOM\_POINT\_NUM; i++) {  
 LatLng randomResult = new LatLng(doubleRandom(input.latitude-randomParam, input.latitude+randomParam),  
 doubleRandom(input.longitude-randomParam, input.longitude+randomParam));  
 poiResult.add(randomResult);  
 }  
 }  
 return poiResult;  
 }  
 private double doubleRandom(double min, double max) {  
 }  
}

**TraceMethod.java**

package com.jnu.lbsprivacy.algorithm;  
import android.annotation.SuppressLint;  
import android.os.Build;  
import androidx.annotation.RequiresApi;  
import com.jnu.lbsprivacy.utils.Path;  
import com.jnu.lbsprivacy.utils.TraceCorrection;  
import com.jnu.lbsprivacy.utils.WSClient;  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Random;  
import io.jenetics.jpx.WayPoint;  
public class TraceMethod {  
 private Path path;  
 private ArrayList<Path> data;  
 final double METER = 0.00001;  
 boolean HTTP\_FLAG = false;  
 public TraceMethod(Path p) {  
 path = p;  
 generate(10,1000,path.getWayPoints());  
 }  
 public void generate(int k,int a,ArrayList<WayPoint> wayPoint)  
 {  
 Random r = new Random();  
 int temp;  
 data = new ArrayList<Path>();  
 data.add(new Path(wayPoint));  
 for(int i = 1;i < k;i++)  
 {  
 data.add(new Path());  
 for(int j=0;j<path.getWayPoints().size();j++)  
 {  
 temp = r.nextInt(a/10)+1;  
 .time(data.get(i-1).getWayPoints().get(j).getTime().get()).build();  
 data.get(i).addWayPoint(temp\_wayPoint);  
 }  
 }  
 }  
 public void generate\_false(int k)  
 {  
 path = new Path();  
 int num = data.get(0).getWayPoints().size()/k;  
 for(int i=0;i<data.get(0).getWayPoints().size();)  
 {  
 Random r = new Random();  
 int temp = r.nextInt(k);  
 if(i+10<data.get(0).getWayPoints().size())  
 {  
 for(int j=0;j<num;j++)  
 {  
 path.addWayPoint(data.get(temp).getoneWayPoint(i));  
 }  
 i += num;  
 }  
 else  
 {  
 path.addWayPoint(data.get(temp).getoneWayPoint(i));  
 i++;  
 }  
 }  
 }  
 @SuppressLint("NewApi")  
 public Path getResult() {  
 Random r = new Random();  
 generate\_false(data.size());  
 return path;  
 }  
}

**TraceRebuiltMethod.java**

package com.jnu.lbsprivacy.algorithm;  
import android.os.Build;  
import android.util.Log;  
import androidx.annotation.RequiresApi;  
import com.baidu.mapapi.model.LatLng;  
import com.jnu.lbsprivacy.utils.DouglasPeucker;  
import com.jnu.lbsprivacy.utils.NavigationTrace;  
import com.jnu.lbsprivacy.utils.Path;  
import com.jnu.lbsprivacy.utils.TraceCorrection;  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
import java.io.BufferedReader;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.net.HttpURLConnection;  
import java.net.URL;  
import java.util.ArrayList;  
import java.util.Random;  
import io.jenetics.jpx.WayPoint;  
public class TraceRebuiltMethod {  
 private Path path;  
 private Path decPath;  
 public int lenResults;  
 int radius;  
 public TraceRebuiltMethod(Path p) {  
 path = p;  
 }  
 private LatLng randomlyChoosePoiByCircle(double lat1,double lng1,int radius){  
 HttpURLConnection connection = null;  
 BufferedReader reader = null;  
 try {  
 URL url = new URL(stringQuery);  
 connection = (HttpURLConnection) url.openConnection();  
 connection.setRequestMethod("GET");  
 connection.setConnectTimeout(50000);  
 connection.setReadTimeout(50000);  
 InputStream in = connection.getInputStream();  
 reader = new BufferedReader(new InputStreamReader(in));  
 StringBuilder result = new StringBuilder();  
 String line;  
 while ((line = reader.readLine()) != null) {  
 result.append(line);  
 }  
 JSONObject jsonObject = new JSONObject(String.valueOf(result));  
 JSONArray jsonArrayResults = jsonObject.getJSONArray("results");  
 lenResults = jsonArrayResults.length();  
 System.out.println("XXXXXX"+lenResults);  
 System.out.println("半径" + radius);  
 if (lenResults == 0){  
 return null;  
 }else{  
 int randIndex = randomlyChoose(lenResults);  
 JSONObject randValue = jsonArrayResults.getJSONObject(randIndex);  
 JSONObject location = randValue.getJSONObject("location");  
 double lat = location.getDouble("lat");  
 double lng = location.getDouble("lng");  
 return new LatLng(lat,lng);  
 }  
 } catch (IOException | JSONException e) {  
 Log.d("eValue", e.toString());  
 e.printStackTrace();  
 } finally {  
 if (reader != null) {  
 try {  
 reader.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 connection.disconnect();  
 }  
 }  
 return null;  
 }  
 private int randomlyChoose(int lenResults) {  
 int x;  
 Random random = new Random();  
 x = random.nextInt(lenResults);  
 return x;  
 }  
 private WayPoint ConvertToWaypoint(LatLng sourceLatLng)  
 {  
 double m=sourceLatLng.latitude;  
 double n=sourceLatLng.longitude;  
 WayPoint wayPoint=WayPoint.builder()  
 .lat(m).lon(n)  
 .build();  
 return wayPoint;  
 }  
 private LatLng ConvertToLatLng(WayPoint wayPoint)  
 {  
 LatLng latLng = new LatLng(wayPoint.getLatitude().doubleValue(),wayPoint.getLongitude().doubleValue());  
 return latLng;  
 }  
 private static WayPoint LatLngCalculateMidPoint(double lat1, double lng1, double lat2, double lng2) {  
 double dLng = Math.toRadians(lng2 - lng1);  
 lat1 = Math.toRadians(lat1);  
 lat2 = Math.toRadians(lat2);  
 lng1 = Math.toRadians(lng1);  
 double lng3 = lng1 + Math.atan2(By, Math.cos(lat1) + Bx);  
 WayPoint midPoint;  
 midPoint = WayPoint.builder()  
 .lat(Math.toDegrees(lat3))  
 .lon(Math.toDegrees(lng3))  
 .build();  
 return midPoint;  
 }  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 private ArrayList<WayPoint> getPoint() {  
 int length = path.getLatLngList().size();  
 if(length>50) {  
 DouglasPeucker douglasPeucker = new DouglasPeucker();  
 decPath = douglasPeucker.run(path, 1e-3d);  
 }  
 else decPath=path;  
 System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*"+decPath.getLatLngList().size());  
 int k = decPath.getLatLngList().size();  
 LatLng latLng;  
 ArrayList<WayPoint> wayPoint = decPath.getWayPoints();  
 ArrayList<WayPoint> mywayPoint = new ArrayList<>();  
 int i = 0;  
 int j;  
 double poilat = 0;  
 double poilng = 0;  
 radius = 200;  
 mywayPoint.add(wayPoint.get(0));  
 for (j = 1; j < k - 2; j++) {  
 double jlat = wayPoint.get(j).getLatitude().doubleValue();  
 double jlng = wayPoint.get(j).getLongitude().doubleValue();  
 double ilat = wayPoint.get(i).getLatitude().doubleValue();  
 double ilng = wayPoint.get(i).getLongitude().doubleValue();  
 WayPoint mid = LatLngCalculateMidPoint(ilat, ilng, wayPoint.get(j).getLatitude().doubleValue(), wayPoint.get(j).getLongitude().doubleValue());  
 latLng = randomlyChoosePoiByCircle(jlat, jlng, radius);  
 if(lenResults>5) radius = radius/2;  
 if(lenResults<5 && radius >3) radius = 200;  
 if(lenResults<3) radius += 200;  
 if (latLng!=null && poilat == latLng.latitude && poilng == latLng.longitude)  
 {  
 latLng = randomlyChoosePoiByCircle(jlat, jlng, radius);  
 mywayPoint.add(ConvertToWaypoint(latLng));  
 poilat = latLng.latitude;  
 poilng = latLng.longitude;  
 }  
 if(latLng==null)  
 {  
 System.out.println("没有找到");  
 i=j;  
 j++;  
 WayPoint mid1 = LatLngCalculateMidPoint( wayPoint.get(i).getLatitude().doubleValue(), wayPoint.get(i).getLongitude().doubleValue(), wayPoint.get(j).getLatitude().doubleValue(), wayPoint.get(j).getLongitude().doubleValue());  
 mywayPoint.add(mid1);  
 }  
 if(latLng!=null && poilat != latLng.latitude && poilng != latLng.longitude)  
 {  
 mywayPoint.add(ConvertToWaypoint(latLng));  
 poilat = latLng.latitude;  
 poilng = latLng.longitude;  
 }  
 i = j;  
 }  
 mywayPoint.add(wayPoint.get(k-1));  
 return mywayPoint;  
 }  
 private WayPoint getMyWaypoint(ArrayList<WayPoint> wayPoints,int k)  
 {  
 return wayPoints.get(k);  
 }  
 public static double getDistance(double lng1,double lat1,double lng2,double lat2){  
 double radLat1 = Math.toRadians(lat1);  
 double radLat2 = Math.toRadians(lat2);  
 double a = radLat1 - radLat2;  
 double b = Math.toRadians(lng1) - Math.toRadians(lng2);  
 return s;  
 }  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 public Path getResult() {  
 Path res = new Path();  
 ArrayList<WayPoint> finelPoint=getPoint();  
 for(int i= 0;i< finelPoint.size();i++) {  
 res.addWayPoint(getMyWaypoint(finelPoint, i));  
 }  
 return res;  
 }  
}

**BaseHook.java**

package com.jnu.lbsprivacy.hooks;  
import android.content.SharedPreferences;  
import de.robv.android.xposed.XC\_MethodHook;  
import de.robv.android.xposed.XSharedPreferences;  
public abstract class BaseHook extends XC\_MethodHook {  
 protected String pkgname;  
 protected ClassLoader classLoader;  
 public BaseHook(int i, ClassLoader classLoader2, String str) {  
 super(i);  
 this.classLoader = classLoader2;  
 this.pkgname = str;  
 }  
 public BaseHook(ClassLoader classLoader2, String str) {  
 this.classLoader = classLoader2;  
 this.pkgname = str;  
 }  
 public abstract void install();  
 public abstract void myAfterHookedMethod(XC\_MethodHook.MethodHookParam methodHookParam);  
 public void afterHookedMethod(XC\_MethodHook.MethodHookParam methodHookParam) {  
 myAfterHookedMethod(methodHookParam);  
 }  
 public abstract String tag();  
 public abstract void myBeforeHookedMethod(XC\_MethodHook.MethodHookParam methodHookParam);  
 public void beforeHookedMethod(XC\_MethodHook.MethodHookParam methodHookParam) {  
 myBeforeHookedMethod(methodHookParam);  
 }  
}

**LMgetBestProviderHook.java**

package com.jnu.lbsprivacy.hooks;  
import android.location.Criteria;  
import android.location.LocationManager;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class LMgetBestProviderHook extends BaseHook{  
 public LMgetBestProviderHook(ClassLoader classLoader2, String str) {  
 super(classLoader2, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod(LocationManager.class, "getBestProvider", Criteria.class, Boolean.TYPE, this);  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log(String.format(tag() + "%s {return GPS\_PROVIDER directly}", this.pkgname));  
 methodHookParam.setResult("gps");  
 }  
 @Override  
 public String tag() {  
 return "LM:gbp";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**LMgetLastKnownLocationHook.java**

package com.jnu.lbsprivacy.hooks;  
import android.location.LocationManager;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.getPrefs;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.setLocationResult;  
public class LMgetLastKnownLocationHook extends BaseHook {  
 public LMgetLastKnownLocationHook(ClassLoader classLoader2, String str) {  
 super(classLoader2, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod(LocationManager.class, "getLastKnownLocation", String.class, this);  
 XposedBridge.log("[LM:glkl] installed");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 setLocationResult(methodHookParam, getPrefs(), pkgname, tag());  
 }  
 @Override  
 public String tag() {  
 return "[LM:glkl]";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**LMgetLastLocationHook.java**

package com.jnu.lbsprivacy.hooks;  
import android.location.LocationManager;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.getPrefs;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.setLocationResult;  
public class LMgetLastLocationHook extends BaseHook {  
 public LMgetLastLocationHook(ClassLoader classLoader2, String str) {  
 super(classLoader2, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod(LocationManager.class, "getLastLocation", this);  
 XposedBridge.log("[LM:gll] installed");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 setLocationResult(methodHookParam, getPrefs(), pkgname, tag());  
 }  
 @Override  
 public String tag() {  
 return "[LM:gll]";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**LMgetProvidersHook.java**

package com.jnu.lbsprivacy.hooks;  
import android.location.LocationManager;  
import java.util.ArrayList;  
import java.util.List;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class LMgetProvidersHook extends BaseHook{  
 public LMgetProvidersHook(ClassLoader classLoader2, String str) {  
 super(classLoader2, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod(LocationManager.class, "getProviders", Boolean.TYPE, this);  
 XposedBridge.log("[LM:gp] installed.");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log("[LM:gp]@ " + this.pkgname + " " + methodHookParam.getResult());  
 Object res, orig;  
 orig = methodHookParam.getResult();  
 if (orig == null) {  
 res = new ArrayList();  
 } else {  
 res = (List) orig;  
 }  
 if (!((List) res).contains("gps")) {  
 XposedBridge.log(String.format(tag() + " {add GPS\_PROVIDER to the list}", this.pkgname));  
 ((List) res).add("gps");  
 }  
 methodHookParam.setResult(res);  
 }  
 @Override  
 public String tag() {  
 return "LM:gp";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**LMisProviderEnableHook.java**

package com.jnu.lbsprivacy.hooks;  
import android.location.Location;  
import android.location.LocationManager;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class LMisProviderEnableHook extends BaseHook {  
 public LMisProviderEnableHook(ClassLoader classLoader2, String str) {  
 super(classLoader2, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod(LocationManager.class, "isProviderEnabled", new Object[]{String.class, this});  
 XposedBridge.log("[LM:ipe] installed.");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log("[LM:ipe]@ " + this.pkgname + " " + methodHookParam.getResult());  
 String orig = (String) methodHookParam.args[0];  
 if ("gps".equals(orig)) {  
 XposedBridge.log(String.format(tag() + " %s {p=%s, gps, %b}", this.pkgname, orig, true));  
 methodHookParam.setResult(true);  
 } else if ("network".equals(orig)) {  
 XposedBridge.log(String.format(tag() + " %s {p=%s, gps, %b}", this.pkgname, orig, true));  
 methodHookParam.setResult(true);  
 }  
 }  
 @Override  
 public String tag() {  
 return "[LM:ipe]";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**LMremoveUpdate.java**

package com.jnu.lbsprivacy.hooks;  
import android.location.LocationListener;  
import android.location.LocationManager;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.removeLocationListener;  
public class LMremoveUpdate extends BaseHook {  
 public LMremoveUpdate(ClassLoader classLoader2, String str) {  
 super(classLoader2, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod(LocationManager.class, "removeUpdates", LocationListener.class, this);  
 XposedBridge.log("[LM:ru] installed");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 }  
 @Override  
 public String tag() {  
 return null;  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log(String.format("[LM:ru] %s {remove proxy}", pkgname));  
 removeLocationListener(methodHookParam, 0);  
 }  
}

**LMrequestLocationUpdateHook.java**

package com.jnu.lbsprivacy.hooks;  
import android.app.PendingIntent;  
import android.content.Context;  
import android.content.Intent;  
import android.location.Location;  
import android.location.LocationListener;  
import android.location.LocationManager;  
import android.os.Looper;  
import com.jnu.lbsprivacy.models.MyLocation;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.addLocationListener;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.getLocation;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.getMyLocation;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.getPrefs;  
public class LMrequestLocationUpdateHook extends BaseHook{  
 public LMrequestLocationUpdateHook(ClassLoader classLoader2, String str) {  
 super(classLoader2, str);  
 }  
 @Override  
 public void install() {  
 Class locationRequestClass = XposedHelpers.findClass("android.location.LocationRequest", this.classLoader);  
 XposedHelpers.findAndHookMethod(LocationManager.class, "requestLocationUpdates", locationRequestClass, LocationListener.class, Looper.class, PendingIntent.class, this);  
 XposedBridge.log("[LM:rlu] installed");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 MyLocation myLocation = getMyLocation(getPrefs());  
 Location location = getLocation(myLocation, true);  
 LocationListener listener = (LocationListener) methodHookParam.args[1];  
 PendingIntent intent = (PendingIntent) methodHookParam.args[3];  
 if (listener != null) {  
 XposedBridge.log(String.format("[LM:rlu] %s {listener, la=%.6f, lo=%.6f, b=%.2f, s=%.2f}", pkgname, myLocation.latLng.latitude, myLocation.latLng.longitude, myLocation.bearing, myLocation.speed));  
 XposedHelpers.callMethod(listener, "onLocationChanged", location);  
 } else if (intent != null) {  
 Intent newIntent = new Intent();  
 newIntent.putExtra("location", new Location(location));  
 Context context = (Context) XposedHelpers.getObjectField(methodHookParam.thisObject, "mContext");  
 if (context != null) {  
 XposedBridge.log(String.format("[LM:rlu] %s {intent, la=%.6f, lo=%.6f, b=%.2f, s=%.2f}", pkgname, myLocation.latLng.latitude, myLocation.latLng.longitude, myLocation.bearing, myLocation.speed));  
 try {  
 intent.send(context, 0, newIntent);  
 } catch (Exception e) {  
 XposedBridge.log(e.toString());  
 }  
 }  
 }  
 }  
 @Override  
 public String tag() {  
 return "[LM:rlu]";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 if (methodHookParam.args[1] != null) {  
 XposedBridge.log("[LM:rlu] Before");  
 addLocationListener(methodHookParam, 1, getPrefs(), pkgname, tag());  
 }  
 }  
}

**TMgetAllCellInfoHook.java**

package com.jnu.lbsprivacy.hooks;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class TMgetAllCellInfoHook extends BaseHook{  
 public TMgetAllCellInfoHook(ClassLoader classLoader, String str) {  
 super(classLoader, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod("android.telephony.TelephonyManager", this.classLoader, "getAllCellInfo", this);  
 XposedBridge.log("[TM:gaci] installed.");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log("[TM:gaci]@ " + this.pkgname + " " + methodHookParam.getResult());  
 XposedBridge.log("[TM:gaci] " + this.pkgname + " {gps=on}");  
 methodHookParam.setResult(null);  
 }  
 @Override  
 public String tag() {  
 return "[TM:gacl]";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**TMgetNeighboringCellInfoHook.java**

package com.jnu.lbsprivacy.hooks;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class TMgetCellLocationHook extends BaseHook{  
 public TMgetCellLocationHook(ClassLoader classLoader, String str) {  
 super(classLoader, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod("android.telephony.TelephonyManager", this.classLoader, "getCellLocation", this);  
 XposedBridge.log("[TM:gcl] installed.");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log("[TM:gcl]@ " + this.pkgname + " " + methodHookParam.getResult());  
 XposedBridge.log("[TM:gcl] " + this.pkgname + " {status(gps)=on}");  
 methodHookParam.setResult(null);  
 }  
 @Override  
 public String tag() {  
 return "TM:gcl";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**TMgetNeighboringCellInfoHook.java**

package com.jnu.lbsprivacy.hooks;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class TMgetNeighboringCellInfoHook extends BaseHook{  
 public TMgetNeighboringCellInfoHook(ClassLoader classLoader, String str) {  
 super(classLoader, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod("android.telephony.TelephonyManager", this.classLoader, "getNeighboringCellInfo", this);  
 XposedBridge.log("[TM:gnci] installed.");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log("[TM:gnci]@ " + this.pkgname + " " + methodHookParam.getResult());  
 XposedBridge.log("[TM:gnci] " + this.pkgname +" {status(gps)=on}");  
 methodHookParam.setResult(null);  
 }  
 @Override  
 public String tag() {  
 return "[TM:gnci]";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**TMgetNetworkOperationHook.java**

package com.jnu.lbsprivacy.hooks;  
import java.util.Locale;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class TMgetNetworkOperationHook extends BaseHook {  
 public TMgetNetworkOperationHook(ClassLoader classLoader, String str) {  
 super(classLoader, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod("android.telephony.TelephonyManager", this.classLoader, "getNetworkOperator", this);  
 XposedBridge.log("[TM:gno] installed.");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log("[TM:gno]@ " + this.pkgname + " " + methodHookParam.getResult());  
 XposedBridge.log("[TM:gno] " + this.pkgname + " {status(gps)=on}");  
 String res = String.format(Locale.US, "%03d", 460);  
 res += String.format(Locale.US, "%02d", 0);  
 methodHookParam.setResult(res);  
 }  
 @Override  
 public String tag() {  
 return "[TM:gno]";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**TMgetPhoneTypeHook.java**

package com.jnu.lbsprivacy.hooks;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class TMgetPhoneTypeHook extends BaseHook {  
 public TMgetPhoneTypeHook(ClassLoader classLoader, String str) {  
 super(classLoader, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod("android.telephony.TelephonyManager", this.classLoader, "getPhoneType", this);  
 XposedBridge.log("[TM:gpt] installed.");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log("[TM:gpt]@ " + this.pkgname + " " + methodHookParam.getResult());  
 XposedBridge.log("[TM:gpt] " + this.pkgname + "{cell(1), phone(1)}");  
 methodHookParam.setResult(1);  
 }  
 @Override  
 public String tag() {  
 return "[TM:gpt]";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**TMgetSimOperatorHook.java**

package com.jnu.lbsprivacy.hooks;  
import java.util.Locale;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class TMgetSimOperatorHook extends BaseHook {  
 public TMgetSimOperatorHook(ClassLoader classLoader, String str) {  
 super(classLoader, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod("android.telephony.TelephonyManager", this.classLoader, "getSimOperator", this);  
 XposedBridge.log("[TM:gso] installed.");  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log("[TM:gso]@ " + this.pkgname + " " + methodHookParam.getResult());  
 String res = String.format(Locale.US, "%03d", 460);  
 res += String.format(Locale.US, "%02d", 0);  
 XposedBridge.log("[TM:gso] " + this.pkgname + " MCC+MNC = " + res);  
 methodHookParam.setResult(res);  
 }  
 @Override  
 public String tag() {  
 return "[TM:gso]";  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
}

**WMgetBSSIDHook.java**

package com.jnu.lbsprivacy.hooks;  
import android.util.Log;  
import de.robv.android.xposed.XC\_MethodHook;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class WMgetBSSIDHook extends BaseHook {  
 public WMgetBSSIDHook(ClassLoader classLoader, String str) {  
 super(classLoader, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod("android.net.wifi.WifiInfo", this.classLoader, "getBSSID", this);  
 XposedBridge.log("[WM:gbss] installed.");  
 }  
 @Override  
 public void myBeforeHookedMethod(XC\_MethodHook.MethodHookParam methodHookParam) {}  
 @Override  
 public void myAfterHookedMethod(XC\_MethodHook.MethodHookParam methodHookParam) {  
 XposedBridge.log("[WM:gbss] " + this.pkgname + " {on, BssId}");  
 methodHookParam.setResult("00:00:00:00:00:00");  
 }  
 @Override  
 public String tag() {  
 return "[WM:gbss]";  
 }  
}

**WMgetScanResultHook.java**

package com.jnu.lbsprivacy.hooks;  
import java.util.ArrayList;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
import de.robv.android.xposed.XposedHelpers;  
public class WMgetScanResultHook extends BaseHook{  
 public WMgetScanResultHook(ClassLoader classLoader, String str) {  
 super(classLoader, str);  
 }  
 @Override  
 public void install() {  
 XposedHelpers.findAndHookMethod("android.net.wifi.WifiManager", this.classLoader, "getScanResults", this);  
 }  
 @Override  
 public void myBeforeHookedMethod(MethodHookParam methodHookParam) {  
 }  
 @Override  
 public void myAfterHookedMethod(MethodHookParam methodHookParam) {  
 XposedBridge.log("[WM:gsr] " + this.pkgname + " {on, wifi}");  
 methodHookParam.setResult(new ArrayList());  
 }  
 @Override  
 public String tag() {  
 return "[WM:gsr]";  
 }  
}

**MyLocation.java**

package com.jnu.lbsprivacy.models;  
import com.baidu.mapapi.model.LatLng;  
public class MyLocation {  
 public float bearing;  
 public float speed;  
 public LatLng latLng;  
 public MyLocation(double latitude, double longitude, float bearing, float speed) {  
 latLng = new LatLng(latitude, longitude);  
 this.bearing = bearing;  
 this.speed = speed;  
 }  
 public MyLocation(LatLng latLng) {  
 this.latLng = latLng;  
 this.bearing = 0.0f;  
 this.speed = 0.0f;  
 }  
 public String toString() {  
 return String.format("MyLocation: {lat=%.6f, lon=%.6f, b=%.2f, s=%.2f}", latLng.latitude, latLng.longitude, bearing, speed);  
 }  
}

**MyLocationListener.java**

package com.jnu.lbsprivacy.models;  
import android.content.SharedPreferences;  
import android.location.Location;  
import android.location.LocationListener;  
import androidx.annotation.NonNull;  
import de.robv.android.xposed.XposedBridge;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.getLocation;  
import static com.jnu.lbsprivacy.utils.LocationHookUtils.getMyLocation;  
public class MyLocationListener implements LocationListener {  
 private int uid;  
 private LocationListener mListener;  
 private String provider;  
 private SharedPreferences sharedPreferences;  
 public MyLocationListener(SharedPreferences sharedPreferences, LocationListener locationListener, String provider, int uid) {  
 this.uid = uid;  
 this.mListener = locationListener;  
 this.provider = provider;  
 this.sharedPreferences = sharedPreferences;  
 }  
 @Override  
 public void onLocationChanged(@NonNull Location location) {  
 MyLocation myLocation = getMyLocation(sharedPreferences);  
 Location newLocation = getLocation(myLocation, true);  
 XposedBridge.log(String.format("[LM:XLL:olc] %s {gps, la=%.6f, lo=%.6f, b=%.2f, s=%.2f}", provider, myLocation.latLng.latitude, myLocation.latLng.longitude, myLocation.bearing, myLocation.speed));  
 this.mListener.onLocationChanged(newLocation);  
 }  
 @Override  
 public void onProviderEnabled(@NonNull String provider) {  
 this.mListener.onProviderEnabled(provider);  
 }  
 @Override  
 public void onProviderDisabled(@NonNull String provider) {  
 this.mListener.onProviderDisabled(provider);  
 }  
}

**UserProfile.java**

package com.jnu.lbsprivacy.models;  
import com.fasterxml.jackson.core.JsonFactory;  
import com.fasterxml.jackson.core.JsonGenerator;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.jnu.lbsprivacy.Config;  
import java.io.IOException;  
import java.io.StringWriter;  
import okhttp3.MediaType;  
import okhttp3.OkHttpClient;  
import okhttp3.Request;  
import okhttp3.RequestBody;  
import okhttp3.Response;  
public class UserProfile {  
 private String uid;  
 public String getUid() {  
 return uid;  
 }  
 public void setUid(String uid) {  
 this.uid = uid;  
 }  
 public int getAge() {  
 return age;  
 }  
 public void setAge(int age) {  
 this.age = age;  
 }  
 public int getPurpose() {  
 return purpose;  
 }  
 public void setPurpose(int purpose) {  
 this.purpose = purpose;  
 }  
 public int getSensitivity() {  
 return sensitivity;  
 }  
 public void setSensitivity(int sensitivity) {  
 this.sensitivity = sensitivity;  
 }  
 public int getLocation() {  
 return location;  
 }  
 public void setLocation(int location) {  
 this.location = location;  
 }  
 public int getTime() {  
 return time;  
 }  
 public void setTime(int time) {  
 this.time = time;  
 }  
 public int getTrust() {  
 return trust;  
 }  
 public void setTrust(int trust) {  
 this.trust = trust;  
 }  
 public int getK() {  
 return k;  
 }  
 public void setK(int k) {  
 this.k = k;  
 }  
 public String getJson() {  
 JsonFactory factory = new JsonFactory();  
 String res = null;  
 StringWriter stringWriter = new StringWriter();  
 try {  
 JsonGenerator generator = factory.createJsonGenerator(stringWriter);  
 generator.useDefaultPrettyPrinter();  
 generator.writeStartObject();  
 generator.writeStringField("uid", uid);  
 generator.writeNumberField("age", age);  
 generator.writeNumberField("purpose", purpose);  
 generator.writeNumberField("sensitivity", sensitivity);  
 generator.writeNumberField("location", location);  
 generator.writeNumberField("time", time);  
 generator.writeNumberField("trust", trust);  
 generator.writeNumberField("k", k);  
 generator.writeEndObject();  
 generator.close();  
 res = stringWriter.toString();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return res;  
 }  
 public String upload() {  
 OkHttpClient client = new OkHttpClient();  
 MediaType JSON = MediaType.parse("application/json; charset=utf-8");  
 RequestBody body = RequestBody.create(JSON, getJson());  
 System.out.println(getJson());  
 Request request = new Request.Builder()  
 .url(Config.API\_URL + "/upload\_profile")  
 .post(body)  
 .build();  
 Response response = null;  
 String resStr = "0";  
 try {  
 response = client.newCall(request).execute();  
 resStr = response.body().string();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return resStr;  
 }  
}

**GalleryFragment.java**

package com.jnu.lbsprivacy.ui.gallery;  
import android.os.Bundle;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.TextView;  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.fragment.app.Fragment;  
import androidx.lifecycle.Observer;  
import androidx.lifecycle.ViewModelProviders;  
import com.jnu.lbsprivacy.R;  
public class GalleryFragment extends Fragment {  
 private GalleryViewModel galleryViewModel;  
 public View onCreateView(@NonNull LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
 galleryViewModel =  
 ViewModelProviders.of(this).get(GalleryViewModel.class);  
 View root = inflater.inflate(R.layout.fragment\_gallery, container, false);  
 final TextView textView = root.findViewById(R.id.text\_gallery);  
 galleryViewModel.getText().observe(getViewLifecycleOwner(), new Observer<String>() {  
 @Override  
 public void onChanged(@Nullable String s) {  
 textView.setText(s);  
 }  
 });  
 return root;  
 }  
}

**GalleryViewModel.java**

package com.jnu.lbsprivacy.ui.gallery;  
import androidx.lifecycle.LiveData;  
import androidx.lifecycle.MutableLiveData;  
import androidx.lifecycle.ViewModel;  
public class GalleryViewModel extends ViewModel {  
 private MutableLiveData<String> mText;  
 public GalleryViewModel() {  
 mText = new MutableLiveData<>();  
 mText.setValue("This is gallery fragment");  
 }  
 public LiveData<String> getText() {  
 return mText;  
 }  
}

**HomeFragment.java**

package com.jnu.lbsprivacy.ui.home;  
import android.os.Bundle;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.TextView;  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.fragment.app.Fragment;  
import androidx.lifecycle.Observer;  
import androidx.lifecycle.ViewModelProviders;  
import com.jnu.lbsprivacy.R;  
public class HomeFragment extends Fragment {  
 private HomeViewModel homeViewModel;  
 public View onCreateView(@NonNull LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
 homeViewModel =  
 ViewModelProviders.of(this).get(HomeViewModel.class);  
 View root = inflater.inflate(R.layout.fragment\_home, container, false);  
 final TextView textView = root.findViewById(R.id.text\_home);  
 homeViewModel.getText().observe(getViewLifecycleOwner(), new Observer<String>() {  
 @Override  
 public void onChanged(@Nullable String s) {  
 textView.setText(s);  
 }  
 });  
 return root;  
 }  
}

**HomeViewModel.java**

package com.jnu.lbsprivacy.ui.home;  
import androidx.lifecycle.LiveData;  
import androidx.lifecycle.MutableLiveData;  
import androidx.lifecycle.ViewModel;  
public class HomeViewModel extends ViewModel {  
 private MutableLiveData<String> mText;  
 public HomeViewModel() {  
 mText = new MutableLiveData<>();  
 mText.setValue("This is home fragment");  
 }  
 public LiveData<String> getText() {  
 return mText;  
 }  
}

**MapFragment.java**

package com.jnu.lbsprivacy.ui.map;  
import android.content.Context;  
import android.content.SharedPreferences;  
import android.graphics.Color;  
import android.os.Build;  
import android.os.Bundle;  
import android.util.Log;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.CompoundButton;  
import android.widget.ImageView;  
import android.widget.Toast;  
import android.widget.ToggleButton;  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.annotation.RequiresApi;  
import androidx.fragment.app.Fragment;  
import androidx.lifecycle.ViewModelProviders;  
import com.baidu.location.BDAbstractLocationListener;  
import com.baidu.location.BDLocation;  
import com.baidu.location.LocationClient;  
import com.baidu.location.LocationClientOption;  
import com.baidu.mapapi.map.BaiduMap;  
import com.baidu.mapapi.map.BitmapDescriptor;  
import com.baidu.mapapi.map.BitmapDescriptorFactory;  
import com.baidu.mapapi.map.InfoWindow;  
import com.baidu.mapapi.map.MapPoi;  
import com.baidu.mapapi.map.MapStatusUpdate;  
import com.baidu.mapapi.map.MapStatusUpdateFactory;  
import com.baidu.mapapi.map.MarkerOptions;  
import com.baidu.mapapi.map.MyLocationConfiguration;  
import com.baidu.mapapi.map.MyLocationData;  
import com.baidu.mapapi.map.OverlayOptions;  
import com.baidu.mapapi.map.PolylineOptions;  
import com.baidu.mapapi.map.TextureMapView;  
import com.baidu.mapapi.model.LatLng;  
import com.baidu.mapapi.search.core.RouteNode;  
import com.baidu.mapapi.search.route.BikingRouteResult;  
import com.baidu.mapapi.search.route.DrivingRouteLine;  
import com.baidu.mapapi.search.route.DrivingRouteResult;  
import com.baidu.mapapi.search.route.IndoorRouteResult;  
import com.baidu.mapapi.search.route.MassTransitRouteResult;  
import com.baidu.mapapi.search.route.OnGetRoutePlanResultListener;  
import com.baidu.mapapi.search.route.TransitRouteResult;  
import com.baidu.mapapi.search.route.WalkingRouteLine;  
import com.baidu.mapapi.search.route.WalkingRouteResult;  
import com.jnu.lbsprivacy.BuildConfig;  
import com.jnu.lbsprivacy.MainActivity;  
import com.jnu.lbsprivacy.R;  
import com.jnu.lbsprivacy.models.MyLocation;  
import com.jnu.lbsprivacy.utils.MultiprocessSharedPreferences;  
import com.jnu.lbsprivacy.utils.NavigationTrace;  
import com.jnu.lbsprivacy.utils.Path;  
import com.jnu.lbsprivacy.algorithm.LocationProtectionMethod;  
import com.jnu.lbsprivacy.utils.TraceCorrection;  
import java.io.File;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.time.ZonedDateTime;  
import java.time.format.DateTimeFormatter;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.List;  
import io.jenetics.jpx.WayPoint;  
public class MapFragment extends Fragment {  
 private static final int NONE\_INPUT = 0;  
 private static final int INJECTION\_LOCATION\_MODE = 1;  
 private static final int INPUT\_TRACE\_MODE\_START = 2;  
 private static final int INPUT\_TRACE\_MODE\_END = 3;  
 private static final int INPUT\_PROTIECTION\_LOCATION\_MODE = 4;  
 private MapViewModel mViewModel;  
 private View mView;  
 private TextureMapView mTextureMapView = null;  
 private LocationClient mLocationClient = null;  
 private LocationClient mRecordLocationClient = null;  
 private BaiduMap mBaiduMap = null;  
 private ImageView mUpdateeLocationBtn;  
 private ToggleButton mRecordBtn;  
 private MyLocationListener myLocationListener = new MyLocationListener();  
 private RecordLocationListener recordLocationListener = new RecordLocationListener();  
 private Path mRecordPath = null;  
 private double lat;  
 private double lon;  
 private Path navTrace;  
 private LatLng startLocation;  
 private LatLng endLocation;  
 private LatLng injectLocation;  
 private boolean isFirstLocation = true;  
 private int inputPointMode;  
 public OnGetRoutePlanResultListener listener = new OnGetRoutePlanResultListener() {  
 @Override  
 public void onGetWalkingRouteResult(WalkingRouteResult walkingRouteResult) {  
 if (walkingRouteResult.getRouteLines().size() > 0) {  
 navTrace =new Path();  
 ArrayList<WayPoint> temp\_list =new ArrayList<WayPoint>();  
 List<WalkingRouteLine.WalkingStep> search = walkingRouteResult.getRouteLines().get(0).getAllStep();  
 Long time = new Date().getTime();  
 for (int i = 0;i < search.size();i++)  
 {  
 List<LatLng> temp = search.get(i).getWayPoints();  
 for(int j= 0;j < temp.size();j++)  
 {  
 WayPoint start;  
 start = WayPoint.builder().lon(temp.get(j).longitude).lat(temp.get(j).latitude).time(time+i\*60000+j\*6000).build();  
 temp\_list.add(start);  
 }  
 }  
 navTrace.setWayPoint(temp\_list);  
 drawNewPath(navTrace);  
 }  
 }  
 @Override  
 public void onGetTransitRouteResult(TransitRouteResult transitRouteResult) {  
 }  
 @Override  
 public void onGetMassTransitRouteResult(MassTransitRouteResult massTransitRouteResult) {  
 }  
 @Override  
 public void onGetDrivingRouteResult(DrivingRouteResult drivingRouteResult) {  
 if (drivingRouteResult.getRouteLines().size() > 0) {  
 navTrace =new Path();  
 ArrayList<WayPoint> temp\_list =new ArrayList<WayPoint>();  
 List<DrivingRouteLine.DrivingStep> search = drivingRouteResult.getRouteLines().get(0).getAllStep();  
 Long time = new Date().getTime();  
 for (int i = 0;i < search.size();i++)  
 {  
 List<LatLng> temp = search.get(i).getWayPoints();  
 for(int j= 0;j < temp.size();j++)  
 {  
 WayPoint start;  
 start = WayPoint.builder().lon(temp.get(j).longitude).lat(temp.get(j).latitude).time(time+i\*60000+j\*6000).build();  
 temp\_list.add(start);  
 }  
 }  
 navTrace.setWayPoint(temp\_list);  
 drawOrigPath(navTrace);  
 saveTrace(navTrace);  
 }  
 }  
 @Override  
 public void onGetIndoorRouteResult(IndoorRouteResult indoorRouteResult) {  
 }  
 @Override  
 public void onGetBikingRouteResult(BikingRouteResult bikingRouteResult) {  
 }  
 };  
 public static MapFragment newInstance() {  
 return new MapFragment();  
 }  
 @Override  
 public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,  
 @Nullable Bundle savedInstanceState) {  
 mView = inflater.inflate(R.layout.map\_fragment, container, false);  
 mTextureMapView = mView.findViewById(R.id.bmapView);  
 mUpdateeLocationBtn = mView.findViewById(R.id.updateLocationBtn);  
 mRecordBtn = mView.findViewById(R.id.recordBtn);  
 mBaiduMap = mTextureMapView.getMap();  
 mBaiduMap.setMyLocationEnabled(true);  
 inputPointMode = NONE\_INPUT;  
 initMap();  
 initRecordLocationClient();  
 return mView;  
 }  
 @Override  
 public void onActivityCreated(@Nullable Bundle savedInstanceState) {  
 super.onActivityCreated(savedInstanceState);  
 Log.d("MapFragment", "onActivityCreated");  
 mUpdateeLocationBtn.setOnClickListener(v -> {  
 mLocationClient.requestLocation();  
 LatLng ll = new LatLng(lat, lon);  
 MapStatusUpdate u = MapStatusUpdateFactory.newLatLng(ll);  
 mBaiduMap.animateMapStatus(u);  
 MapStatusUpdate u1 = MapStatusUpdateFactory.zoomTo(15f);  
 mBaiduMap.animateMapStatus(u1);  
 }  
 );  
 mRecordBtn.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {  
 @Override  
 public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {  
 if (isChecked) {  
 Log.d("MapFragment", "Start Recording");  
 mRecordPath = new Path();  
 mRecordLocationClient.start();  
 } else{  
 Log.d("MapFragment", "Stop Recording");  
 mRecordLocationClient.stop();  
 File externalDir = getContext().getExternalFilesDir("GPX");  
 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy\_MM\_dd\_HH\_mm\_ss");  
 String filename = ZonedDateTime.now().format(formatter) + ".gpx";  
 File file = new File(externalDir + "/" + filename);  
 try {  
 FileOutputStream outputStream = new FileOutputStream(file);  
 Toast.makeText(getContext(), "New route save to: " + filename, Toast.LENGTH\_SHORT).show();  
 mRecordPath.saveGPX(outputStream);  
 }  
 catch (IOException e) {  
 Log.d("MapFragment", e.toString());  
 }  
 Log.d("MapFragment", mRecordPath.toString());  
 }  
 }  
 });  
 mViewModel = ViewModelProviders.of(this).get(MapViewModel.class);  
 }  
 @Override  
 public void onResume() {  
 super.onResume();  
 Log.d("MapFragment", "onResume");  
 mTextureMapView.onResume();  
 }  
 @Override  
 public void onPause() {  
 super.onPause();  
 Log.d("MapFragment", "onPause");  
 mTextureMapView.onPause();  
 }  
 @Override  
 public void onDestroy() {  
 super.onDestroy();  
 Log.d("MapFragment", "onDestroy");  
 mLocationClient.unRegisterLocationListener(myLocationListener);  
 mLocationClient.stop();  
 mRecordLocationClient.unRegisterLocationListener(recordLocationListener);  
 mRecordLocationClient.stop();  
 mBaiduMap.setMyLocationEnabled(false);  
 mTextureMapView.onDestroy();  
 mTextureMapView = null;  
 }  
 public class MyLocationListener extends BDAbstractLocationListener {  
 @Override  
 public void onReceiveLocation(BDLocation location) {  
 if (location == null || mTextureMapView == null) {  
 return;  
 }  
 lat = location.getLatitude();  
 lon = location.getLongitude();  
 Log.d("latlon", "" + lat + "," + lon);  
 MyLocationData locData = new MyLocationData.Builder()  
 .accuracy(location.getRadius())  
 .direction(location.getDirection())  
 .latitude(location.getLatitude())  
 .longitude(location.getLongitude()).build();  
 if (isFirstLocation) {  
 isFirstLocation = false;  
 mBaiduMap.setMyLocationData(locData);  
 MapStatusUpdate u = MapStatusUpdateFactory.zoomTo(15f);  
 mBaiduMap.animateMapStatus(u);  
 }  
 }  
 }  
 public class RecordLocationListener extends BDAbstractLocationListener {  
 @Override  
 public void onReceiveLocation(BDLocation location) {  
 if (location == null || mTextureMapView == null) {  
 return;  
 }  
 lat = location.getLatitude();  
 lon = location.getLongitude();  
 Log.d("latlon", "" + lat + "," + lon);  
 MyLocationData locData = new MyLocationData.Builder()  
 .accuracy(location.getRadius())  
 .direction(location.getDirection())  
 .latitude(location.getLatitude())  
 .longitude(location.getLongitude()).build();  
 mRecordPath.addWayPoint(WayPoint.builder().lat(lat).lon(lon).time(ZonedDateTime.now()).build());  
 Log.d("RecordLocation", locData.latitude + ", " + locData.longitude);  
 }  
 }  
 public class MyOnClickListener implements BaiduMap.OnMapClickListener {  
 @Override  
 public void onMapClick(LatLng point) {  
 long startTime, endTime;  
 double runtTime;  
 String text;  
 Log.d("OnClick", point.toString());  
 if (inputPointMode == INJECTION\_LOCATION\_MODE) {  
 injectLocation = point;  
 addCurrentMarker(point);  
 injectLocation(new MyLocation(point));  
 setInjectionModeOff();  
 } else if (inputPointMode == INPUT\_TRACE\_MODE\_START) {  
 startLocation = point;  
 addCurrentMarker(point);  
 inputPointMode = INPUT\_TRACE\_MODE\_END;  
 } else if (inputPointMode == INPUT\_TRACE\_MODE\_END) {  
 endLocation = point;  
 addCurrentMarker(point);  
 inputPointMode = NONE\_INPUT;  
 drawNavTrace();  
 }  
 else if (inputPointMode == INPUT\_PROTIECTION\_LOCATION\_MODE) {  
 clearOverlay();  
 startTime = System.currentTimeMillis();  
 addCurrentMarker(point);  
 LocationProtectionMethod m = new LocationProtectionMethod(point);  
 endTime = System.currentTimeMillis();  
 runtTime = (endTime - startTime) / 1000.0;  
 text = String.format("加载位置点时长：%.3fs", runtTime);  
 Toast.makeText(getActivity(), text, Toast.LENGTH\_SHORT).show();  
 ArrayList<LatLng> results = m.getResult();  
 for (LatLng p: results) {  
 addFakeMarker(p);  
 }  
 setLocationProtectOff();  
 }  
 }  
 @Override  
 public void onMapPoiClick(MapPoi mapPoi) {  
 Log.d("OnClick", mapPoi.toString());  
 }  
 }  
 private void initMap() {  
 mLocationClient = new LocationClient(getActivity().getApplicationContext());  
 LocationClientOption option = new LocationClientOption();  
 option.setOpenGps(true);  
 option.setCoorType("bd09ll");  
 option.setScanSpan(1000);  
 option.setEnableSimulateGps(true);  
 option.setLocationMode(LocationClientOption.LocationMode.Hight\_Accuracy);  
 mLocationClient.setLocOption(option);  
 mLocationClient.registerLocationListener(myLocationListener);  
 mLocationClient.start();  
 MyLocationConfiguration myLocationConfiguration = new MyLocationConfiguration(MyLocationConfiguration.LocationMode.FOLLOWING, true, (BitmapDescriptor) null);  
 myLocationConfiguration.accuracyCircleFillColor = 0x00000000;  
 myLocationConfiguration.accuracyCircleStrokeColor = 0x00000000;  
 mBaiduMap.setMyLocationConfiguration(myLocationConfiguration);  
 MyOnClickListener myOnClickListener = new MyOnClickListener();  
 mBaiduMap.setOnMapClickListener(myOnClickListener);  
 }  
 public void pathcorr(Path path)  
 {  
 TraceCorrection corr = new TraceCorrection(path);  
 corr.correct\_Nav(listener);  
 }  
 private void initRecordLocationClient() {  
 mRecordLocationClient = new LocationClient(getActivity().getApplicationContext());  
 LocationClientOption option = new LocationClientOption();  
 option.setOpenGps(true);  
 option.setCoorType("gcj02");  
 option.setEnableSimulateGps(true);  
 option.setLocationMode(LocationClientOption.LocationMode.Hight\_Accuracy);  
 mRecordLocationClient.setLocOption(option);  
 mRecordLocationClient.registerLocationListener(recordLocationListener);  
 }  
 public void addCurrentMarker(LatLng point) {  
 BitmapDescriptor bitmap = BitmapDescriptorFactory.fromResource(R.drawable.location\_here1);  
 OverlayOptions option = new MarkerOptions().position(point).icon(bitmap);  
 mBaiduMap.addOverlay(option);  
 }  
 public void addFakeMarker(LatLng point) {  
 BitmapDescriptor bitmap = BitmapDescriptorFactory.fromResource(R.drawable.location\_fake1);  
 OverlayOptions option = new MarkerOptions().position(point).icon(bitmap);  
 mBaiduMap.addOverlay(option);  
 }  
 public void setNavTrace() {  
 Toast.makeText(getActivity(), "请选择路径起点", Toast.LENGTH\_SHORT).show();  
 clearOverlay();  
 inputPointMode = INPUT\_TRACE\_MODE\_START;  
 }  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 public void drawNavTrace() {  
 NavigationTrace nv = new NavigationTrace();  
 nv.setpoint(startLocation,endLocation);  
 nv.getResult(listener);  
 }  
 public void saveTrace(Path navTrace) {  
 File externalDir = getContext().getExternalFilesDir("GPX");  
 DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy\_MM\_dd\_HH\_mm\_ss");  
 String filename = ZonedDateTime.now().format(formatter) + ".gpx";  
 File file = new File(externalDir + "/" + filename);  
 try {  
 FileOutputStream outputStream = new FileOutputStream(file);  
 Toast.makeText(getContext(), "New route save to: " + filename, Toast.LENGTH\_SHORT).show();  
 navTrace.saveGPX(outputStream);  
 }  
 catch (IOException e) {  
 Log.d("MapFragment", e.toString());  
 }  
 }  
 public void setInjectionModeOn() {  
 Log.d("MapFragment", "Injection mode on");  
 Toast.makeText(getActivity(), "请选择注入点", Toast.LENGTH\_SHORT).show();  
 clearOverlay();  
 inputPointMode = INJECTION\_LOCATION\_MODE;  
 }  
 public void setInjectionModeOff() {  
 Log.d("MapFragment", "注入点输入完成");  
 inputPointMode = NONE\_INPUT;  
 }  
 public void setLocationProtectOn() {  
 Log.d("MapFragment", "Injection mode on");  
 Toast.makeText(getActivity(), "请选择保护的位置", Toast.LENGTH\_SHORT).show();  
 clearOverlay();  
 inputPointMode = INPUT\_PROTIECTION\_LOCATION\_MODE;  
 }  
 public void setLocationProtectOff() {  
 Log.d("MapFragment", "保护点输入完成");  
 inputPointMode = NONE\_INPUT;  
 }  
 public LatLng getInjectionPoint() {  
 return injectLocation;  
 }  
 public void drawNewPath(Path path) {  
 OverlayOptions polylineOption = new PolylineOptions().points(path.getoralLatLngList())  
 .width(5).color(Color.BLUE);  
 mBaiduMap.addOverlay(polylineOption);  
 }  
 public void drawOrigPath(Path path) {  
 OverlayOptions polylineOption = new PolylineOptions().points(path.getoralLatLngList())  
 .width(10).color(Color.RED);  
 mBaiduMap.addOverlay(polylineOption);  
 }  
 public Path getNavTrace()  
 {  
 return navTrace;  
 }  
 public void clearOverlay() {  
 mBaiduMap.clear();  
 }  
 public LatLng getCurrentLocation() {  
 return new LatLng(lat, lon);  
 }  
 public Path getmRecordPath() {  
 return mRecordPath;  
 }  
 public void injectLocation(MyLocation myLocation) {  
 SharedPreferences sharedPref = MultiprocessSharedPreferences.getSharedPreferences(getActivity(), BuildConfig.APPLICATION\_ID, Context.MODE\_PRIVATE);  
 SharedPreferences.Editor editor = sharedPref.edit();  
 editor.putLong("latitude", Double.doubleToLongBits(myLocation.latLng.latitude));  
 editor.putLong("longitude", Double.doubleToLongBits(myLocation.latLng.longitude));  
 editor.putFloat("bearing", myLocation.bearing);  
 editor.putFloat("float", myLocation.speed);  
 editor.apply();  
 Toast.makeText(getActivity(), String.format("注入位置：（%.5f, %.5f）", myLocation.latLng.latitude, myLocation.latLng.longitude), Toast.LENGTH\_LONG).show();  
 }  
}

**MapViewModel.java**

package com.jnu.lbsprivacy.ui.map;  
import androidx.lifecycle.ViewModel;  
public class MapViewModel extends ViewModel {  
}

**SlideshowFragment.java**

package com.jnu.lbsprivacy.ui.slideshow;  
import android.os.Bundle;  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.TextView;  
import androidx.annotation.NonNull;  
import androidx.annotation.Nullable;  
import androidx.fragment.app.Fragment;  
import androidx.lifecycle.Observer;  
import androidx.lifecycle.ViewModelProviders;  
import com.jnu.lbsprivacy.R;  
public class SlideshowFragment extends Fragment {  
 private SlideshowViewModel slideshowViewModel;  
 public View onCreateView(@NonNull LayoutInflater inflater,  
 ViewGroup container, Bundle savedInstanceState) {  
 slideshowViewModel =  
 ViewModelProviders.of(this).get(SlideshowViewModel.class);  
 View root = inflater.inflate(R.layout.fragment\_slideshow, container, false);  
 final TextView textView = root.findViewById(R.id.text\_slideshow);  
 slideshowViewModel.getText().observe(getViewLifecycleOwner(), new Observer<String>() {  
 @Override  
 public void onChanged(@Nullable String s) {  
 textView.setText(s);  
 }  
 });  
 return root;  
 }  
}

**SlideshowViewModel.java**

package com.jnu.lbsprivacy.ui.slideshow;  
import androidx.lifecycle.LiveData;  
import androidx.lifecycle.MutableLiveData;  
import androidx.lifecycle.ViewModel;  
public class SlideshowViewModel extends ViewModel {  
 private MutableLiveData<String> mText;  
 public SlideshowViewModel() {  
 mText = new MutableLiveData<>();  
 mText.setValue("This is slideshow fragment");  
 }  
 public LiveData<String> getText() {  
 return mText;  
 }  
}

**DouglasPeucker.java**

package com.jnu.lbsprivacy.utils;  
import android.util.Log;  
import java.lang.reflect.Array;  
import java.util.ArrayList;  
import java.util.List;  
import io.jenetics.jpx.WayPoint;  
public class DouglasPeucker {  
 public DouglasPeucker() { }  
 private static final double sqr(double x) {  
 return Math.pow(x, 2);  
 }  
 private static final double distanceBetweenPoints(double vx, double vy, double wx, double wy) {  
 return sqr(vx - wx) + sqr(vy - wy);  
 }  
 private static final double distanceToSegmentSquared(double px, double py, double vx, double vy, double wx, double wy) {  
 final double l2 = distanceBetweenPoints(vx, vy, wx, wy);  
 if (l2 == 0)  
 return distanceBetweenPoints(px, py, vx, vy);  
 if (t < 0)  
 return distanceBetweenPoints(px, py, vx, vy);  
 if (t > 1)  
 return distanceBetweenPoints(px, py, wx, wy);  
 }  
 private static final double perpendicularDistance(double px, double py, double vx, double vy, double wx, double wy) {  
 return Math.sqrt(distanceToSegmentSquared(px, py, vx, vy, wx, wy));  
 }  
 private static final void douglasPeucker(Path list, int s, int e, double epsilon, Path resultList) {  
 double dmax = 0;  
 int index = 0;  
 final int start = s;  
 final int end = e-1;  
 for (int i=start+1; i<end; i++) {  
 final double px = list.getoneWayPoint(i).getLatitude().doubleValue();  
 final double py = list.getoneWayPoint(i).getLongitude().doubleValue();  
 final double vx = list.getoneWayPoint(start).getLatitude().doubleValue();  
 final double vy = list.getoneWayPoint(start).getLongitude().doubleValue();  
 final double wx = list.getoneWayPoint(end).getLatitude().doubleValue();  
 final double wy = list.getoneWayPoint(end).getLongitude().doubleValue();  
 final double d = perpendicularDistance(px, py, vx, vy, wx, wy);  
 if (d > dmax) {  
 index = i;  
 dmax = d;  
 }  
 }  
 if (dmax > epsilon) {  
 douglasPeucker(list, s, index, epsilon, resultList);  
 douglasPeucker(list, index, e, epsilon, resultList);  
 } else {  
 if ((end-start)>0) {  
 resultList.addWayPoint(list.getWayPoints().get(start));  
 resultList.addWayPoint(list.getWayPoints().get(end));  
 } else {  
 resultList.addWayPoint(list.getWayPoints().get(start));  
 }  
 }  
 }  
 public static Path run(Path list, double epsilon) {  
 if(list.getWayPoints().size() <= 40)  
 return list;  
 Log.d("before DouglasPeucker", String.valueOf(list.getWayPoints().size()));  
 Path resultList = new Path();  
 douglasPeucker(list, 0, list.getWayPoints().size(), epsilon, resultList);  
 if(resultList.getWayPoints().size() <=20) {  
 resultList = new Path();  
 douglasPeucker(list, 0, list.getWayPoints().size(), epsilon/2, resultList);  
 }  
 Log.d("after DouglasPeucker", String.valueOf(resultList.getWayPoints().size()));  
 return resultList;  
 }  
 public static ArrayList<Path> runMulti(ArrayList<Path> lists, double epsilon) {  
 ArrayList<Path> resultLists = new ArrayList<>();  
 StringBuilder beforeTag = new StringBuilder();  
 StringBuilder afterTag = new StringBuilder();  
 for(Path list:lists) {  
 beforeTag.append(list.getWayPoints().size());  
 beforeTag.append(" ");  
 if(list.getWayPoints().size() <= 40)  
 resultLists.add(list);  
 else {  
 Path resultList = new Path();  
 douglasPeucker(list, 0, list.getWayPoints().size(), epsilon, resultList);  
 if(resultList.getWayPoints().size() <=20) {  
 resultList = new Path();  
 douglasPeucker(list, 0, list.getWayPoints().size(), epsilon/2, resultList);  
 }  
 resultLists.add(resultList);  
 }  
 }  
 for(Path list:resultLists) {  
 afterTag.append(list.getWayPoints().size());  
 afterTag.append(" ");  
 }  
 Log.d("before DouglasPeucker", beforeTag.toString());  
 Log.d("after DouglasPeucker", afterTag.toString());  
 return resultLists;  
 }  
}

**Grid.java**

package com.jnu.lbsprivacy.utils;  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.Random;  
import io.jenetics.jpx.Latitude;  
import io.jenetics.jpx.Longitude;  
import io.jenetics.jpx.WayPoint;  
public class Grid {  
 private double latMin;  
 private double latMax;  
 private double lngMin;  
 private double lngMax;  
 private int count;  
 public Grid(double \_latMin, double \_latMax, double \_lngMin, double \_lngMax) {  
 latMin = \_latMin;  
 latMax = \_latMax;  
 lngMin = \_lngMin;  
 lngMax = \_lngMax;  
 }  
 public Grid(Path p) {  
 ArrayList<WayPoint> wayPoints = p.getWayPoints();  
 if (wayPoints == null || wayPoints.size() == 0) {  
 throw new ExceptionInInitializerError();  
 }  
 latMin = wayPoints.stream()  
 .map(WayPoint::getLatitude)  
 .map(Latitude::doubleValue)  
 .min(Comparator.comparing(Double::doubleValue)).get();  
 latMax = wayPoints.stream()  
 .map(WayPoint::getLatitude)  
 .map(Latitude::doubleValue)  
 .max(Comparator.comparing(Double::doubleValue)).get();  
 lngMin = wayPoints.stream()  
 .map(WayPoint::getLongitude)  
 .map(Longitude::doubleValue)  
 .min(Comparator.comparing(Double::doubleValue)).get();  
 lngMax = wayPoints.stream()  
 .map(WayPoint::getLongitude)  
 .map(Longitude::doubleValue)  
 .max(Comparator.comparing(Double::doubleValue)).get();  
 }  
 public double getLatMin() {  
 return latMin;  
 }  
 public double getLatMax() {  
 return latMax;  
 }  
 public double getLngMin() {  
 return lngMin;  
 }  
 public double getLngMax() {  
 return lngMax;  
 }  
 public void setCount(int count) {  
 this.count = count;  
 }  
 public int getCount() {  
 return count;  
 }  
 public boolean isWayPointInside(WayPoint wayPoint) {  
 return wayPoint.getLatitude().doubleValue() >= latMin  
 && wayPoint.getLatitude().doubleValue() <= latMax  
 && wayPoint.getLongitude().doubleValue() >= lngMin  
 && wayPoint.getLongitude().doubleValue() <= lngMax;  
 }  
 public WayPoint getRandomWayPointInside() {  
 Random r = new Random();  
 return WayPoint.builder().lat(lat).lon(lng).build();  
 }  
 @Override  
 public String toString() {  
 return "Grid{" +  
 "latMin=" + latMin +  
 ", latMax=" + latMax +  
 ", lngMin=" + lngMin +  
 ", lngMax=" + lngMax +  
 ", count=" + count +  
 '}';  
 }  
}

**LocationConverter.java**

package com.jnu.lbsprivacy.utils;  
import com.baidu.mapapi.model.LatLng;  
public class LocationConverter {  
 private static final double LAT\_OFFSET\_0(double x, double y) {  
 }  
 private static final double LAT\_OFFSET\_1(double x, double y) {  
 }  
 private static final double LAT\_OFFSET\_2(double x, double y) {  
 }  
 private static final double LAT\_OFFSET\_3(double x, double y) {  
 }  
 private static final double LON\_OFFSET\_0(double x, double y) {  
 }  
 private static final double LON\_OFFSET\_1(double x, double y) {  
 }  
 private static final double LON\_OFFSET\_2(double x, double y) {  
 }  
 private static final double LON\_OFFSET\_3(double x, double y) {  
 }  
 private static double RANGE\_LON\_MAX = 137.8347;  
 private static double RANGE\_LON\_MIN = 72.004;  
 private static double RANGE\_LAT\_MAX = 55.8271;  
 private static double RANGE\_LAT\_MIN = 0.8293;  
 private static double jzA = 6378245.0;  
 private static double jzEE = 0.00669342162296594323;  
 public static double transformLat(double x, double y) {  
 double ret = LAT\_OFFSET\_0(x, y);  
 ret += LAT\_OFFSET\_1(x, y);  
 ret += LAT\_OFFSET\_2(x, y);  
 ret += LAT\_OFFSET\_3(x, y);  
 return ret;  
 }  
 public static double transformLon(double x, double y) {  
 double ret = LON\_OFFSET\_0(x, y);  
 ret += LON\_OFFSET\_1(x, y);  
 ret += LON\_OFFSET\_2(x, y);  
 ret += LON\_OFFSET\_3(x, y);  
 return ret;  
 }  
 public static boolean outOfChina(double lat, double lon) {  
 if (lon < RANGE\_LON\_MIN || lon > RANGE\_LON\_MAX)  
 return true;  
 if (lat < RANGE\_LAT\_MIN || lat > RANGE\_LAT\_MAX)  
 return true;  
 return false;  
 }  
 public static LatLng gcj02Encrypt(double ggLat, double ggLon) {  
 double mgLat;  
 double mgLon;  
 if (outOfChina(ggLat, ggLon)) {  
 return new LatLng(ggLat, ggLon);  
 }  
 double dLat = transformLat(ggLon - 105.0, ggLat - 35.0);  
 double dLon = transformLon(ggLon - 105.0, ggLat - 35.0);  
 double magic = Math.sin(radLat);  
 double sqrtMagic = Math.sqrt(magic);  
 mgLat = ggLat + dLat;  
 mgLon = ggLon + dLon;  
 return new LatLng(mgLat, mgLon);  
 }  
 public static LatLng gcj02Decrypt(double gjLat, double gjLon) {  
 LatLng gPt = gcj02Encrypt(gjLat, gjLon);  
 double dLon = gPt.longitude - gjLon;  
 double dLat = gPt.latitude - gjLat;  
 return new LatLng(gjLat - dLat, gjLon - dLon);  
 }  
 public static LatLng bd09Decrypt(double bdLat, double bdLon) {  
 double x = bdLon - 0.0065, y = bdLat - 0.006;  
 }  
 public static LatLng bd09Encrypt(double ggLat, double ggLon) {  
 double x = ggLon, y = ggLat;  
 return new LatLng(bdPt\_latitude, bdPt\_longitude);  
 }  
 public static LatLng wgs84ToGcj02(LatLng location) {  
 return gcj02Encrypt(location.latitude, location.longitude);  
 }  
 public static LatLng gcj02ToWgs84(LatLng location) {  
 return gcj02Decrypt(location.latitude, location.longitude);  
 }  
 public static LatLng wgs84ToBd09(LatLng location) {  
 LatLng gcj02Pt = gcj02Encrypt(location.latitude, location.longitude);  
 return bd09Encrypt(gcj02Pt.latitude, gcj02Pt.longitude);  
 }  
 public static LatLng gcj02ToBd09(LatLng location) {  
 return bd09Encrypt(location.latitude, location.longitude);  
 }  
 public static LatLng bd09ToGcj02(LatLng location) {  
 return bd09Decrypt(location.latitude, location.longitude);  
 }  
 public static LatLng bd09ToWgs84(LatLng location) {  
 LatLng gcj02 = bd09ToGcj02(location);  
 return gcj02Decrypt(gcj02.latitude, gcj02.longitude);  
 }  
}

**LocationHookUtils.java**

package com.jnu.lbsprivacy.utils;  
import android.app.AndroidAppHelper;  
import android.content.Context;  
import android.content.SharedPreferences;  
import android.location.GpsStatus;  
import android.location.Location;  
import android.location.LocationListener;  
import android.os.Binder;  
import android.os.SystemClock;  
import com.baidu.mapapi.model.LatLng;  
import com.jnu.lbsprivacy.BuildConfig;  
import com.jnu.lbsprivacy.models.MyLocation;  
import com.jnu.lbsprivacy.models.MyLocationListener;  
import java.util.Map;  
import java.util.WeakHashMap;  
import de.robv.android.xposed.XC\_MethodHook;  
import de.robv.android.xposed.XSharedPreferences;  
import de.robv.android.xposed.XposedBridge;  
public class LocationHookUtils {  
 public static final Map<Object, Object> locationListenerMap = new WeakHashMap();  
 public static SharedPreferences getPrefs() {  
 Context context = (Context) AndroidAppHelper.currentApplication();  
 MultiprocessSharedPreferences.setAuthority("com.jnu.lbsprivacy.provider");  
 return MultiprocessSharedPreferences.getSharedPreferences(context, BuildConfig.APPLICATION\_ID, context.MODE\_PRIVATE);  
 }  
 public static void addLocationListener(XC\_MethodHook.MethodHookParam methodHookParam, int idx, SharedPreferences sharedPreferences, String pkgname, String tag) {  
 Object listener = methodHookParam.args[idx];  
 synchronized (locationListenerMap) {  
 if (locationListenerMap.containsKey(listener)) {  
 XposedBridge.log(tag + String.format(" %s {reuse proxy}", pkgname));  
 } else if (locationListenerMap.containsValue(listener)) {  
 XposedBridge.log(tag + String.format(" %s {already proxy}", pkgname));  
 } else {  
 XposedBridge.log(tag + String.format(" %s {creating proxy}", pkgname));  
 MyLocationListener mylistener = new MyLocationListener(sharedPreferences, (LocationListener) methodHookParam.args[idx], pkgname, Binder.getCallingUid());  
 synchronized (locationListenerMap) {  
 locationListenerMap.put(listener, mylistener);  
 }  
 methodHookParam.args[idx] = mylistener;  
 }  
 }  
 }  
 public static void removeLocationListener(XC\_MethodHook.MethodHookParam methodHookParam, int idx) {  
 if (methodHookParam.args.length > idx && methodHookParam.args[idx] != null) {  
 Object listener = methodHookParam.args[idx];  
 synchronized (listener) {  
 if (locationListenerMap.containsKey(listener)) {  
 methodHookParam.args[idx] = locationListenerMap.remove(listener);  
 }  
 }  
 }  
 }  
 public static Location getLocation(MyLocation myLocation, Boolean isConvert) {  
 Location location = new Location("gps");  
 if (isConvert) {  
 myLocation.latLng = LocationConverter.bd09ToWgs84(myLocation.latLng);  
 }  
 location.setLatitude(myLocation.latLng.latitude);  
 location.setLongitude(myLocation.latLng.longitude);  
 location.setTime(System.currentTimeMillis());  
 location.setElapsedRealtimeNanos(SystemClock.elapsedRealtimeNanos());  
 location.setAccuracy(100.0f);  
 location.setBearing(myLocation.bearing);  
 location.setSpeed(myLocation.speed);  
 return location;  
 }  
 public static MyLocation getMyLocation(SharedPreferences SharedPreferences) {  
 double latitude = Double.longBitsToDouble(SharedPreferences.getLong("latitude", Double.doubleToLongBits(0.0d)));  
 double longitude = Double.longBitsToDouble(SharedPreferences.getLong("longitude", Double.doubleToLongBits(0.0d)));  
 float bearing = SharedPreferences.getFloat("bearing", 0.0f);  
 float speed = SharedPreferences.getFloat("speed", 0.0f);  
 return new MyLocation(latitude, longitude, bearing, speed);  
 }  
 public static void setLocationResult(XC\_MethodHook.MethodHookParam methodHookParam, SharedPreferences sharedPreferences, String tag, String pkgname) {  
 Location orig = (Location) methodHookParam.getResult();  
 XposedBridge.log(String.format(tag + "@ %s {on, la=%.6f, lo=%.6f, b=%.2f, s=%.2f}", pkgname, orig.getLatitude(), orig.getLongitude(), orig.getBearing(), orig.getSpeed()));  
 MyLocation myLocation = getMyLocation(sharedPreferences);  
 Location location = getLocation(myLocation, true);  
 XposedBridge.log(String.format(tag + " %s {on, la=%.6f, lo=%.6f, b=%.2f, s=%.2f}", pkgname, myLocation.latLng.longitude, myLocation.latLng.latitude, myLocation.latLng.longitude, myLocation.bearing, myLocation.speed));  
 methodHookParam.setResult(location);  
 }  
}

**MultiprocessSharedPreferences.java**

package com.jnu.lbsprivacy.utils;  
import android.content.BroadcastReceiver;  
import android.content.ContentProvider;  
import android.content.ContentResolver;  
import android.content.ContentValues;  
import android.content.Context;  
import android.content.Intent;  
import android.content.IntentFilter;  
import android.content.SharedPreferences;  
import android.content.UriMatcher;  
import android.content.pm.PackageInfo;  
import android.content.pm.PackageManager;  
import android.content.pm.ProviderInfo;  
import android.database.Cursor;  
import android.database.MatrixCursor;  
import android.net.Uri;  
import android.os.Build;  
import android.os.Bundle;  
import android.os.DeadObjectException;  
import androidx.annotation.NonNull;  
import android.util.Log;  
import com.jnu.lbsprivacy.BuildConfig;  
import java.lang.reflect.Constructor;  
import java.lang.reflect.Field;  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
import java.util.ArrayList;  
import java.util.Arrays;  
import java.util.HashMap;  
import java.util.HashSet;  
import java.util.List;  
import java.util.Map;  
import java.util.Set;  
import java.util.WeakHashMap;  
public class MultiprocessSharedPreferences extends ContentProvider implements SharedPreferences {  
 private static final String TAG = "MultiprocessSharedPreferences";  
 public static final boolean DEBUG = BuildConfig.DEBUG;  
 private Context mContext;  
 private String mName;  
 private int mMode;  
 private boolean mIsSafeMode;  
 private static final Object CONTENT = new Object();  
 private WeakHashMap<OnSharedPreferenceChangeListener, Object> mListeners;  
 private BroadcastReceiver mReceiver;  
 private static String AUTHORITY;  
 private static volatile Uri AUTHORITY\_URI;  
 private UriMatcher mUriMatcher;  
 private static final String KEY = "value";  
 private static final String KEY\_NAME = "name";  
 private static final String PATH\_WILDCARD = "\*/";  
 private static final String PATH\_GET\_ALL = "getAll";  
 private static final String PATH\_GET\_STRING = "getString";  
 private static final String PATH\_GET\_INT = "getInt";  
 private static final String PATH\_GET\_LONG = "getLong";  
 private static final String PATH\_GET\_FLOAT = "getFloat";  
 private static final String PATH\_GET\_BOOLEAN = "getBoolean";  
 private static final String PATH\_CONTAINS = "contains";  
 private static final String PATH\_APPLY = "apply";  
 private static final String PATH\_COMMIT = "commit";  
 private static final String PATH\_REGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER = "registerOnSharedPreferenceChangeListener";  
 private static final String PATH\_UNREGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER = "unregisterOnSharedPreferenceChangeListener";  
 private static final String PATH\_GET\_STRING\_SET = "getStringSet";  
 private static final int GET\_ALL = 1;  
 private static final int GET\_STRING = 2;  
 private static final int GET\_INT = 3;  
 private static final int GET\_LONG = 4;  
 private static final int GET\_FLOAT = 5;  
 private static final int GET\_BOOLEAN = 6;  
 private static final int CONTAINS = 7;  
 private static final int APPLY = 8;  
 private static final int COMMIT = 9;  
 private static final int REGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER = 10;  
 private static final int UNREGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER = 11;  
 private static final int GET\_STRING\_SET = 12;  
 private HashMap<String, Integer> mListenersCount;  
 private static class ReflectionUtil {  
 public static ContentValues contentValuesNewInstance(HashMap<String, Object> values) {  
 try {  
 c.setAccessible(true);  
 return c.newInstance(values);  
 } catch (IllegalArgumentException e) {  
 throw new RuntimeException(e);  
 } catch (IllegalAccessException e) {  
 throw new RuntimeException(e);  
 } catch (InvocationTargetException e) {  
 throw new RuntimeException(e);  
 } catch (NoSuchMethodException e) {  
 throw new RuntimeException(e);  
 } catch (InstantiationException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 public static Editor editorPutStringSet(Editor editor, String key, Set<String> values) {  
 try {  
 return (Editor) method.invoke(editor, key, values);  
 } catch (IllegalArgumentException e) {  
 throw new RuntimeException(e);  
 } catch (IllegalAccessException e) {  
 throw new RuntimeException(e);  
 } catch (InvocationTargetException e) {  
 throw new RuntimeException(e);  
 } catch (NoSuchMethodException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 @SuppressWarnings("unchecked")  
 public static Set<String> sharedPreferencesGetStringSet(SharedPreferences sharedPreferences, String key, Set<String> values) {  
 try {  
 return (Set<String>) method.invoke(sharedPreferences, key, values);  
 } catch (IllegalArgumentException e) {  
 throw new RuntimeException(e);  
 } catch (IllegalAccessException e) {  
 throw new RuntimeException(e);  
 } catch (InvocationTargetException e) {  
 throw new RuntimeException(e);  
 } catch (NoSuchMethodException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 public static void editorApply(Editor editor) {  
 try {  
 method.invoke(editor);  
 } catch (IllegalArgumentException e) {  
 throw new RuntimeException(e);  
 } catch (IllegalAccessException e) {  
 throw new RuntimeException(e);  
 } catch (InvocationTargetException e) {  
 throw new RuntimeException(e);  
 } catch (NoSuchMethodException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 public static String contentProvidermAuthority(ContentProvider contentProvider) {  
 try {  
 mAuthority.setAccessible(true);  
 return (String) mAuthority.get(contentProvider);  
 } catch (NoSuchFieldException e) {  
 throw new RuntimeException(e);  
 } catch (IllegalAccessException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
 private boolean isSafeMode(Context context) {  
 boolean isSafeMode = false;  
 try {  
 isSafeMode = context.getPackageManager().isSafeMode();  
 } catch (RuntimeException e) {  
 if (!isPackageManagerHasDiedException(e)) {  
 throw e;  
 }  
 }  
 return isSafeMode;  
 }  
 public static void setAuthority(String authority) {  
 AUTHORITY = authority;  
 }  
 private boolean checkInitAuthority(Context context) {  
 if (AUTHORITY\_URI == null) {  
 synchronized (MultiprocessSharedPreferences.this) {  
 if (AUTHORITY\_URI == null) {  
 if(AUTHORITY == null) {  
 if (Build.VERSION.SDK\_INT >= 21 && this instanceof ContentProvider) {  
 AUTHORITY = ReflectionUtil.contentProvidermAuthority(this);  
 } else {  
 PackageInfo packageInfos = null;  
 try {  
 packageInfos = context.getPackageManager().getPackageInfo(context.getPackageName(), PackageManager.GET\_PROVIDERS);  
 } catch (PackageManager.NameNotFoundException e) {  
 if (DEBUG) {  
 e.printStackTrace();  
 }  
 } catch (RuntimeException e) {  
 if (!isPackageManagerHasDiedException(e)) {  
 throw new RuntimeException("checkInitAuthority", e);  
 }  
 }  
 if (packageInfos != null && packageInfos.providers != null) {  
 for (ProviderInfo providerInfo : packageInfos.providers) {  
 if (providerInfo.name.equals(MultiprocessSharedPreferences.class.getName())) {  
 AUTHORITY = providerInfo.authority;  
 break;  
 }  
 }  
 }  
 }  
 }  
 if (DEBUG) {  
 if (AUTHORITY == null) {  
 throw new IllegalArgumentException("'AUTHORITY' initialize failed, Unable to find explicit provider class " + MultiprocessSharedPreferences.class.getName() + "; have you declared this provider in your AndroidManifest.xml?");  
 } else {  
 Log.d(TAG, "checkInitAuthority.AUTHORITY = " + AUTHORITY);  
 }  
 }  
 }  
 }  
 }  
 return AUTHORITY\_URI != null;  
 }  
 private boolean isPackageManagerHasDiedException(Throwable e) {  
 if (e instanceof RuntimeException  
 && e.getMessage() != null  
 && e.getMessage().contains("Package manager has died")) {  
 Throwable cause = getLastCause(e);  
 if (cause instanceof DeadObjectException || cause.getClass().getName().equals("android.os.TransactionTooLargeException")) {  
 return true;  
 }  
 }  
 return false;  
 }  
 private boolean isUnstableCountException(Throwable e) {  
 if (e instanceof RuntimeException  
 && e.getMessage() != null  
 && e.getMessage().contains("unstableCount < 0: -1")) {  
 if (getLastCause(e) instanceof IllegalStateException) {  
 return true;  
 }  
 }  
 return false;  
 }  
 private Throwable getLastCause(Throwable tr) {  
 Throwable cause = tr.getCause();  
 Throwable causeLast = null;  
 while (cause != null) {  
 causeLast = cause;  
 cause = cause.getCause();  
 }  
 if (causeLast == null) {  
 causeLast = new Throwable();  
 }  
 return causeLast;  
 }  
 public static SharedPreferences getSharedPreferences(Context context, String name, int mode) {  
 return new MultiprocessSharedPreferences(context, name, mode);  
 }  
 @Deprecated  
 public MultiprocessSharedPreferences() {  
 }  
 private MultiprocessSharedPreferences(Context context, String name, int mode) {  
 mContext = context;  
 mName = name;  
 mMode = mode;  
 mIsSafeMode = isSafeMode(mContext);  
 }  
 @SuppressWarnings("unchecked")  
 @Override  
 public Map<String, ?> getAll() {  
 Map<String, ?> value = (Map<String, ?>) getValue(PATH\_GET\_ALL, null, null);  
 return value == null ? new HashMap<String, Object>() : value;  
 }  
 @Override  
 public String getString(String key, String defValue) {  
 return (String) getValue(PATH\_GET\_STRING, key, defValue);  
 }  
 @SuppressWarnings("unchecked")  
 public Set<String> getStringSet(String key, Set<String> defValues) {  
 return (Set<String>) getValue(PATH\_GET\_STRING\_SET, key, defValues);  
 }  
 @Override  
 public int getInt(String key, int defValue) {  
 return (Integer) getValue(PATH\_GET\_INT, key, defValue);  
 }  
 @Override  
 public long getLong(String key, long defValue) {  
 return (Long) getValue(PATH\_GET\_LONG, key, defValue);  
 }  
 @Override  
 public float getFloat(String key, float defValue) {  
 return (Float) getValue(PATH\_GET\_FLOAT, key, defValue);  
 }  
 @Override  
 public boolean getBoolean(String key, boolean defValue) {  
 return (Boolean) getValue(PATH\_GET\_BOOLEAN, key, defValue);  
 }  
 @Override  
 public boolean contains(String key) {  
 return (Boolean) getValue(PATH\_CONTAINS, key, false);  
 }  
 @Override  
 public Editor edit() {  
 return new EditorImpl();  
 }  
 @Override  
 public void registerOnSharedPreferenceChangeListener(OnSharedPreferenceChangeListener listener) {  
 synchronized (this) {  
 if (mListeners == null) {  
 mListeners = new WeakHashMap<OnSharedPreferenceChangeListener, Object>();  
 }  
 Boolean result = (Boolean) getValue(PATH\_REGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER, null, false);  
 if (result != null && result) {  
 mListeners.put(listener, CONTENT);  
 if (mReceiver == null) {  
 mReceiver = new BroadcastReceiver() {  
 @Override  
 public void onReceive(Context context, Intent intent) {  
 String name = intent.getStringExtra(KEY\_NAME);  
 @SuppressWarnings("unchecked")  
 List<String> keysModified = (List<String>) intent.getSerializableExtra(KEY);  
 if (mName.equals(name) && keysModified != null) {  
 Set<OnSharedPreferenceChangeListener> listeners = new HashSet<OnSharedPreferenceChangeListener>(mListeners.keySet());  
 for (int i = keysModified.size() - 1; i >= 0; i--) {  
 final String key = keysModified.get(i);  
 for (OnSharedPreferenceChangeListener listener : listeners) {  
 if (listener != null) {  
 listener.onSharedPreferenceChanged(MultiprocessSharedPreferences.this, key);  
 }  
 }  
 }  
 }  
 }  
 };  
 mContext.registerReceiver(mReceiver, new IntentFilter(makeAction(mName)));  
 }  
 }  
 }  
 }  
 @Override  
 public void unregisterOnSharedPreferenceChangeListener(OnSharedPreferenceChangeListener listener) {  
 synchronized (this) {  
 if (mListeners != null) {  
 mListeners.remove(listener);  
 if (mListeners.isEmpty() && mReceiver != null) {  
 mContext.unregisterReceiver(mReceiver);  
 }  
 }  
 }  
 }  
 public final class EditorImpl implements Editor {  
 private final Map<String, Object> mModified = new HashMap<String, Object>();  
 private boolean mClear = false;  
 @Override  
 public Editor putString(String key, String value) {  
 synchronized (this) {  
 mModified.put(key, value);  
 return this;  
 }  
 }  
 public Editor putStringSet(String key, Set<String> values) {  
 synchronized (this) {  
 mModified.put(key, (values == null) ? null : new HashSet<String>(values));  
 return this;  
 }  
 }  
 @Override  
 public Editor putInt(String key, int value) {  
 synchronized (this) {  
 mModified.put(key, value);  
 return this;  
 }  
 }  
 @Override  
 public Editor putLong(String key, long value) {  
 synchronized (this) {  
 mModified.put(key, value);  
 return this;  
 }  
 }  
 @Override  
 public Editor putFloat(String key, float value) {  
 synchronized (this) {  
 mModified.put(key, value);  
 return this;  
 }  
 }  
 @Override  
 public Editor putBoolean(String key, boolean value) {  
 synchronized (this) {  
 mModified.put(key, value);  
 return this;  
 }  
 }  
 @Override  
 public Editor remove(String key) {  
 synchronized (this) {  
 mModified.put(key, null);  
 return this;  
 }  
 }  
 @Override  
 public Editor clear() {  
 synchronized (this) {  
 mClear = true;  
 return this;  
 }  
 }  
 @Override  
 public void apply() {  
 setValue(PATH\_APPLY);  
 }  
 @Override  
 public boolean commit() {  
 return setValue(PATH\_COMMIT);  
 }  
 private boolean setValue(String pathSegment) {  
 boolean result = false;  
 String[] selectionArgs = new String[] { String.valueOf(mMode), String.valueOf(mClear) };  
 synchronized (this) {  
 Uri uri = Uri.withAppendedPath(Uri.withAppendedPath(AUTHORITY\_URI, mName), pathSegment);  
 ContentValues values = ReflectionUtil.contentValuesNewInstance((HashMap<String, Object>) mModified);  
 try {  
 result = mContext.getContentResolver().update(uri, values, null, selectionArgs) > 0;  
 } catch (IllegalArgumentException e) {  
 if (DEBUG) {  
 e.printStackTrace();  
 }  
 } catch (RuntimeException e) {  
 if (!isPackageManagerHasDiedException(e) && !isUnstableCountException(e)) {  
 throw new RuntimeException(e);  
 }  
 }  
 }  
 }  
 if (DEBUG) {  
 Log.d(TAG, "setValue.mName = " + mName + ", pathSegment = " + pathSegment + ", mModified.size() = " + mModified.size());  
 }  
 return result;  
 }  
 }  
 private Object getValue(String pathSegment, String key, Object defValue) {  
 Object v = null;  
 Uri uri = Uri.withAppendedPath(Uri.withAppendedPath(AUTHORITY\_URI, mName), pathSegment);  
 String[] projection = null;  
 if (PATH\_GET\_STRING\_SET.equals(pathSegment) && defValue != null) {  
 @SuppressWarnings("unchecked")  
 Set<String> set = (Set<String>) defValue;  
 projection = new String[set.size()];  
 set.toArray(projection);  
 }  
 String[] selectionArgs = new String[] { String.valueOf(mMode), key, defValue == null ? null : String.valueOf(defValue) };  
 Cursor cursor = null;  
 try {  
 cursor = mContext.getContentResolver().query(uri, projection, null, selectionArgs, null);  
 } catch (SecurityException e) {  
 if (DEBUG) {  
 e.printStackTrace();  
 }  
 } catch (RuntimeException e) {  
 if (!isPackageManagerHasDiedException(e) && !isUnstableCountException(e)) {  
 throw new RuntimeException(e);  
 }  
 }  
 if (cursor != null) {  
 Bundle bundle = null;  
 try {  
 bundle = cursor.getExtras();  
 } catch (RuntimeException e) {  
 if (DEBUG) {  
 e.printStackTrace();  
 }  
 }  
 if (bundle != null) {  
 v = bundle.get(KEY);  
 bundle.clear();  
 }  
 cursor.close();  
 }  
 }  
 if (DEBUG) {  
 Log.d(TAG, "getValue.mName = " + mName + ", pathSegment = " + pathSegment + ", key = " + key + ", defValue = " + defValue);  
 }  
 return v == null ? defValue : v;  
 }  
 private String makeAction(String name) {  
 return String.format("%1$s\_%2$s", MultiprocessSharedPreferences.class.getName(), name);  
 }  
 @Override  
 public boolean onCreate() {  
 if (checkInitAuthority(getContext())) {  
 mUriMatcher = new UriMatcher(UriMatcher.NO\_MATCH);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_GET\_ALL, GET\_ALL);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_GET\_STRING, GET\_STRING);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_GET\_INT, GET\_INT);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_GET\_LONG, GET\_LONG);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_GET\_FLOAT, GET\_FLOAT);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_GET\_BOOLEAN, GET\_BOOLEAN);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_CONTAINS, CONTAINS);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_APPLY, APPLY);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_COMMIT, COMMIT);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_REGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER, REGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_UNREGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER, UNREGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER);  
 mUriMatcher.addURI(AUTHORITY, PATH\_WILDCARD + PATH\_GET\_STRING\_SET, GET\_STRING\_SET);  
 return true;  
 } else {  
 return false;  
 }  
 }  
 @Override  
 public Cursor query(@NonNull Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder) {  
 String name = uri.getPathSegments().get(0);  
 int mode = Integer.parseInt(selectionArgs[0]);  
 String key = selectionArgs[1];  
 String defValue = selectionArgs[2];  
 Bundle bundle = new Bundle();  
 switch (mUriMatcher.match(uri)) {  
 case GET\_ALL:  
 bundle.putSerializable(KEY, (HashMap<String, ?>) getSystemSharedPreferences(name, mode).getAll());  
 break;  
 case GET\_STRING:  
 bundle.putString(KEY, getSystemSharedPreferences(name, mode).getString(key, defValue));  
 break;  
 case GET\_INT:  
 bundle.putInt(KEY, getSystemSharedPreferences(name, mode).getInt(key, Integer.parseInt(defValue)));  
 break;  
 case GET\_LONG:  
 bundle.putLong(KEY, getSystemSharedPreferences(name, mode).getLong(key, Long.parseLong(defValue)));  
 break;  
 case GET\_FLOAT:  
 bundle.putFloat(KEY, getSystemSharedPreferences(name, mode).getFloat(key, Float.parseFloat(defValue)));  
 break;  
 case GET\_BOOLEAN:  
 bundle.putBoolean(KEY, getSystemSharedPreferences(name, mode).getBoolean(key, Boolean.parseBoolean(defValue)));  
 break;  
 case CONTAINS:  
 bundle.putBoolean(KEY, getSystemSharedPreferences(name, mode).contains(key));  
 break;  
 case REGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER: {  
 checkInitListenersCount();  
 Integer countInteger = mListenersCount.get(name);  
 int count = (countInteger == null ? 0 : countInteger) + 1;  
 mListenersCount.put(name, count);  
 countInteger = mListenersCount.get(name);  
 bundle.putBoolean(KEY, count == (countInteger == null ? 0 : countInteger));  
 }  
 break;  
 case UNREGISTER\_ON\_SHARED\_PREFERENCE\_CHANGE\_LISTENER: {  
 checkInitListenersCount();  
 Integer countInteger = mListenersCount.get(name);  
 int count = (countInteger == null ? 0 : countInteger) - 1;  
 if (count <= 0) {  
 mListenersCount.remove(name);  
 bundle.putBoolean(KEY, !mListenersCount.containsKey(name));  
 } else {  
 mListenersCount.put(name, count);  
 countInteger = mListenersCount.get(name);  
 bundle.putBoolean(KEY, count == (countInteger == null ? 0 : countInteger));  
 }  
 }  
 break;  
 case GET\_STRING\_SET: {  
 Set<String> set = null;  
 if (projection != null) {  
 set = new HashSet<String>(Arrays.asList(projection));  
 }  
 bundle.putSerializable(KEY, (HashSet<String>) ReflectionUtil.sharedPreferencesGetStringSet(getSystemSharedPreferences(name, mode), key, set));  
 }  
 }  
 default:  
 if (DEBUG) {  
 throw new IllegalArgumentException("At query, This is Unknown Uri：" + uri + ", AUTHORITY = " + AUTHORITY);  
 }  
 }  
 return new BundleCursor(bundle);  
 }  
 @SuppressWarnings("unchecked")  
 @Override  
 public int update(@NonNull Uri uri, ContentValues values, String selection, String[] selectionArgs) {  
 int result = 0;  
 String name = uri.getPathSegments().get(0);  
 int mode = Integer.parseInt(selectionArgs[0]);  
 SharedPreferences preferences = getSystemSharedPreferences(name, mode);  
 int match = mUriMatcher.match(uri);  
 switch (match) {  
 case APPLY:  
 case COMMIT:  
 boolean hasListeners = mListenersCount != null && mListenersCount.get(name) != null && mListenersCount.get(name) > 0;  
 ArrayList<String> keysModified = null;  
 Map<String, Object> map = null;  
 if (hasListeners) {  
 keysModified = new ArrayList<String>();  
 map = (Map<String, Object>) preferences.getAll();  
 }  
 Editor editor = preferences.edit();  
 boolean clear = Boolean.parseBoolean(selectionArgs[1]);  
 if (clear) {  
 if (hasListeners && !map.isEmpty()) {  
 for (Map.Entry<String, Object> entry : map.entrySet()) {  
 keysModified.add(entry.getKey());  
 }  
 }  
 editor.clear();  
 }  
 for (Map.Entry<String, Object> entry : values.valueSet()) {  
 String k = entry.getKey();  
 Object v = entry.getValue();  
 if (v instanceof EditorImpl || v == null) {  
 editor.remove(k);  
 if (hasListeners && map.containsKey(k)) {  
 keysModified.add(k);  
 }  
 } else {  
 if (hasListeners && (!map.containsKey(k) || (map.containsKey(k) && !v.equals(map.get(k))))) {  
 keysModified.add(k);  
 }  
 }  
 if (v instanceof String) {  
 editor.putString(k, (String) v);  
 } else if (v instanceof Set) {  
 } else if (v instanceof Integer) {  
 editor.putInt(k, (Integer) v);  
 } else if (v instanceof Long) {  
 editor.putLong(k, (Long) v);  
 } else if (v instanceof Float) {  
 editor.putFloat(k, (Float) v);  
 } else if (v instanceof Boolean) {  
 editor.putBoolean(k, (Boolean) v);  
 }  
 }  
 if (hasListeners && keysModified.isEmpty()) {  
 result = 1;  
 } else {  
 switch (match) {  
 case APPLY:  
 result = 1;  
 notifyListeners(name, keysModified);  
 break;  
 case COMMIT:  
 if (editor.commit()) {  
 result = 1;  
 notifyListeners(name, keysModified);  
 }  
 break;  
 default:  
 break;  
 }  
 }  
 values.clear();  
 break;  
 default:  
 if (DEBUG) {  
 throw new IllegalArgumentException("At update, This is Unknown Uri：" + uri + ", AUTHORITY = " + AUTHORITY);  
 }  
 }  
 return result;  
 }  
 @Override  
 public String getType(@NonNull Uri uri) {  
 throw new UnsupportedOperationException("No external call");  
 }  
 @Override  
 public Uri insert(@NonNull Uri uri, ContentValues values) {  
 throw new UnsupportedOperationException("No external insert");  
 }  
 @Override  
 public int delete(@NonNull Uri uri, String selection, String[] selectionArgs) {  
 throw new UnsupportedOperationException("No external delete");  
 }  
 private SharedPreferences getSystemSharedPreferences(String name, int mode) {  
 return getContext().getSharedPreferences(name, mode);  
 }  
 private void checkInitListenersCount() {  
 if (mListenersCount == null) {  
 mListenersCount = new HashMap<String, Integer>();  
 }  
 }  
 private void notifyListeners(String name, ArrayList<String> keysModified) {  
 if (keysModified != null && !keysModified.isEmpty()) {  
 Intent intent = new Intent();  
 intent.setAction(makeAction(name));  
 intent.setPackage(getContext().getPackageName());  
 intent.putExtra(KEY\_NAME, name);  
 intent.putExtra(KEY, keysModified);  
 getContext().sendBroadcast(intent);  
 }  
 }  
 private static final class BundleCursor extends MatrixCursor {  
 private Bundle mBundle;  
 public BundleCursor(Bundle extras) {  
 super(new String[] {}, 0);  
 mBundle = extras;  
 }  
 @Override  
 public Bundle getExtras() {  
 return mBundle;  
 }  
 @Override  
 public Bundle respond(Bundle extras) {  
 mBundle = extras;  
 return mBundle;  
 }  
 }  
}

**NavigationTrace.java**

package com.jnu.lbsprivacy.utils;  
import android.os.Build;  
import androidx.annotation.RequiresApi;  
import com.baidu.mapapi.model.LatLng;  
import com.baidu.mapapi.search.route.DrivingRoutePlanOption;  
import com.baidu.mapapi.search.route.OnGetRoutePlanResultListener;  
import com.baidu.mapapi.search.route.PlanNode;  
import com.baidu.mapapi.search.route.RoutePlanSearch;  
import io.jenetics.jpx.WayPoint;  
public class NavigationTrace {  
 private Path path;  
 private LatLng startLocation;  
 private LatLng endLocation;  
 private boolean FLAG;  
 private OnGetRoutePlanResultListener listener;  
 public void setpoint(LatLng start, LatLng end)  
 {  
 startLocation = start;  
 endLocation = end;  
 }  
 public void selectway() {  
 RoutePlanSearch mSearch = RoutePlanSearch.newInstance();  
 mSearch.setOnGetRoutePlanResultListener(listener);  
 PlanNode stNode = PlanNode.withLocation(startLocation);  
 PlanNode enNode = PlanNode.withLocation(endLocation);  
 mSearch.drivingSearch((new DrivingRoutePlanOption())  
 .from(stNode)  
 .to(enNode));  
 }  
 public Path getResult(OnGetRoutePlanResultListener listen) {  
 listener = listen;  
 selectway();  
 return path;  
 }  
}

**Path.java**  
package com.jnu.lbsprivacy.utils;  
import android.util.Log;  
import com.baidu.mapapi.model.LatLng;  
import com.baidu.mapapi.utils.CoordinateConverter;  
import com.fasterxml.jackson.core.JsonFactory;  
import com.fasterxml.jackson.core.JsonGenerator;  
import com.jnu.lbsprivacy.Config;  
import java.io.FileOutputStream;  
import java.io.IOException;  
import java.io.InputStream;  
import java.io.StringWriter;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
import io.jenetics.jpx.GPX;  
import io.jenetics.jpx.WayPoint;  
import okhttp3.MediaType;  
import okhttp3.OkHttpClient;  
import okhttp3.Request;  
import okhttp3.RequestBody;  
import okhttp3.Response;  
public class Path {  
 private List<WayPoint> wayPoints;  
 public Path() {  
 wayPoints = new ArrayList<WayPoint>();  
 }  
 public Path(ArrayList<WayPoint> \_wayPoints) {  
 wayPoints = \_wayPoints;  
 }  
 public void loadFromGPX(InputStream inputStream) {  
 try {  
 wayPoints = GPX.read(inputStream)  
 .wayPoints().collect(Collectors.toList());  
 } catch (IOException e) {  
 Log.d("Path", e.toString());  
 }  
 }  
 public ArrayList<LatLng> getLatLngList() {  
 ArrayList<LatLng> res = new ArrayList<LatLng>();  
 CoordinateConverter converter = new CoordinateConverter().from(CoordinateConverter.CoordType.COMMON);  
 for (WayPoint waypoint : wayPoints) {  
 LatLng commonCoord = new LatLng(waypoint.getLatitude().doubleValue(), waypoint.getLongitude().doubleValue());  
 LatLng baiduCoord = converter.coord(commonCoord).convert();  
 res.add(baiduCoord);  
 }  
 return res;  
 }  
 public ArrayList<LatLng> getoralLatLngList() {  
 ArrayList<LatLng> res = new ArrayList<LatLng>();  
 for (WayPoint waypoint : wayPoints) {  
 LatLng commonCoord = new LatLng(waypoint.getLatitude().doubleValue(), waypoint.getLongitude().doubleValue());  
 res.add(commonCoord);  
 }  
 return res;  
 }  
 public void addWayPoint(WayPoint wp) {  
 wayPoints.add(wp);  
 }  
 public ArrayList<WayPoint> getWayPoints() {  
 return (ArrayList<WayPoint>) wayPoints;  
 }  
 public void setWayPoint(ArrayList<WayPoint> wayPoint) {  
 this.wayPoints = wayPoint;  
 }  
 public void saveGPX(FileOutputStream outputStream) {  
 GPX.Builder gpxBuiler = GPX.builder();  
 for (WayPoint waypoint : wayPoints) {  
 gpxBuiler.addWayPoint(waypoint);  
 }  
 try {  
 GPX.write(gpxBuiler.build(), outputStream);  
 } catch (IOException e) {  
 Log.d("Path", e.toString());  
 }  
 }  
 public WayPoint getoneWayPoint(int k) {  
 WayPoint temp;  
 temp = WayPoint.builder().lon(wayPoints.get(k).getLongitude())  
 .lat(wayPoints.get(k).getLatitude()).time(wayPoints.get(k).getTime().get()).build();  
 return temp;  
 }  
 @Override  
 public String toString() {  
 return "Path{" +  
 "wayPoints=" + wayPoints +  
 '}';  
 }  
 public String getJson() {  
 JsonFactory factory = new JsonFactory();  
 String res = null;  
 StringWriter stringWriter = new StringWriter();  
 try {  
 JsonGenerator generator = factory.createJsonGenerator(stringWriter);  
 generator.useDefaultPrettyPrinter();  
 generator.writeStartObject();  
 generator.writeArrayFieldStart("path");  
 for (WayPoint p : wayPoints) {  
 generator.writeStartObject();  
 generator.writeNumberField("lat", p.getLatitude().doubleValue());  
 generator.writeNumberField("lon", p.getLongitude().doubleValue());  
 generator.writeStringField("time", p.getTime().get().toString());  
 generator.writeEndObject();  
 }  
 generator.writeEndArray();  
 generator.writeEndObject();  
 generator.close();  
 res = stringWriter.toString();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return res;  
 }  
 public void upload() {  
 OkHttpClient client = new OkHttpClient();  
 MediaType JSON = MediaType.parse("application/json");  
 RequestBody body = RequestBody.create(JSON, getJson());  
 Request request = new Request.Builder()  
 .url(Config.API\_URL + "/uptrace")  
 .post(body)  
 .build();  
 Response response = null;  
 try {  
 response = client.newCall(request).execute();  
 String resStr = response.body().string();  
 System.out.println(resStr);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**TraceCorrection.java**

package com.jnu.lbsprivacy.utils;  
import android.annotation.SuppressLint;  
import android.os.Build;  
import androidx.annotation.RequiresApi;  
import com.baidu.mapapi.model.LatLng;  
import com.baidu.mapapi.search.route.DrivingRoutePlanOption;  
import com.baidu.mapapi.search.route.OnGetRoutePlanResultListener;  
import com.baidu.mapapi.search.route.PlanNode;  
import com.baidu.mapapi.search.route.RoutePlanSearch;  
import com.baidu.mapapi.search.route.WalkingRoutePlanOption;  
import org.json.JSONArray;  
import org.json.JSONException;  
import org.json.JSONObject;  
import java.util.ArrayList;  
import java.util.Date;  
import java.util.HashMap;  
import java.util.Map;  
import io.jenetics.jpx.WayPoint;  
public class TraceCorrection {  
 private Path path;  
 private LatLng startLocation;  
 private LatLng endLocation;  
 private int pointlen = 50;  
 private OnGetRoutePlanResultListener listener;  
 public TraceCorrection()  
 {  
 path = new Path();  
 }  
 @RequiresApi(api = Build.VERSION\_CODES.O)  
 public TraceCorrection(Path newpath)  
 {  
 path = newpath;  
 }  
 public void setpoint(LatLng start, LatLng end)  
 {  
 startLocation = start;  
 endLocation = end;  
 }  
 public void selectway() {  
 RoutePlanSearch mSearch = RoutePlanSearch.newInstance();  
 mSearch.setOnGetRoutePlanResultListener(listener);  
 System.out.println(startLocation.latitude +"->"+endLocation.latitude);  
 System.out.println(startLocation.longitude +"->"+endLocation.longitude);  
 PlanNode stNode = PlanNode.withLocation(startLocation);  
 PlanNode enNode = PlanNode.withLocation(endLocation);  
 mSearch.walkingSearch(new WalkingRoutePlanOption()  
 .from(stNode)  
 .to(enNode));  
 }  
 public void correct\_Nav(OnGetRoutePlanResultListener listen)  
 {  
 listener = listen;  
 ArrayList<WayPoint> waypoint = new ArrayList<WayPoint>();  
 ArrayList<WayPoint> temp;  
 if(path.getWayPoints().size()<pointlen)pointlen=path.getWayPoints().size();  
 int pointSize = path.getWayPoints().size()/pointlen;  
 if (pointSize==0) pointSize=1;  
 for(int i = 0;i<pointlen;i++)  
 {  
 if(i==pointlen-1)  
 {  
 LatLng start = new LatLng(path.getWayPoints().get(i\*pointSize).getLatitude().doubleValue(), path.getWayPoints().get(i\*pointSize).getLongitude().doubleValue());  
 LatLng end = new LatLng(path.getWayPoints().get(path.getWayPoints().size()-1).getLatitude().doubleValue(), path.getWayPoints().get(path.getWayPoints().size()-1).getLongitude().doubleValue());  
 if(start.longitude==end.longitude&&start.latitude==end.latitude) continue;  
 setpoint(start,end);  
 }  
 else  
 {  
 LatLng start = new LatLng(path.getWayPoints().get(i\*pointSize).getLatitude().doubleValue(), path.getWayPoints().get(i\*pointSize).getLongitude().doubleValue());  
 LatLng end = new LatLng(path.getWayPoints().get((i+1)\*pointSize).getLatitude().doubleValue(), path.getWayPoints().get((i+1)\*pointSize).getLongitude().doubleValue());  
 if(start.longitude==end.longitude&&start.latitude==end.latitude) continue;  
 setpoint(start,end);  
 }  
 selectway();  
 }  
 }  
 public Path getresult()  
 {  
 return path;  
 }  
}