Il progetto: la *APP*

File *manifest*

Oltre alla *activity* principale che realizza l'interfaccia con l'utente e attiva o meno l'accesso ai dati di localizzazione del dispositivo GPS avviando il *service* illustrato nel paragrafo precedente, altre due *activity* secondarie permettono di visualizzare rispettivamente il percorso sovrapposto a una mappa e di mostrare le informazioni relative al percorso effettuato.

Nel file *manifest* della *APP* devono essere specificati i permessi hardware e software da richiedere al momento dell'installazione; nel caso specifico questa *APP* richiede l'accesso alla posizione del dispositivo rilevata dal sensore GPS o da altri servizi di localizzazione:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Il metodo più semplice che una *APP* Android può impiegare per visualizzare dati su una mappa consiste nell'accedere ai servizi web resi disponibili da Google *online* utilizzando il *Google Play Services* SDK. Non è necessario specificare nel file *manifest* tutti i permessi richiesti dalla libreria, poiché includerla permette di inserire automaticamente anche i file *manifest* contenuti al suo interno, con i relativi permessi.

Una volta importata la libreria delle API per i servizi *Google Play* e ottenuta l'autorizzazione per tali servizi mediante una API *key*, è necessario specificare tale valore all'interno del file *manifest* in un *tag meta-data* dell'elemento *application*

```
<meta-data
  android:name="it.google.android.geo.API_KEY"
  android:value="..............................."/>
```

dove l'attributo *value* deve essere valorizzato la chiave ottenuta[1].

Non è invece richiesto nessun permesso per scrivere i dati raccolti su file, perché la *APP* usa la memoria interna a sua disposizione, e nessun permesso per l'accesso alla rete Internet in quanto ereditato dalla libreria *Google Play Services*.

Il file *manifest* dovrebbe quindi essere simile al seguente:

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

  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

  <application
    android:name=".GeoTrackerApplication"
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
```

SDK per i servizi Google

Google Play Services SDK è una libreria esterna che estende il *Software Development Kit* standard del sistema operativo Android.
Essa contiene numerose API per accedere ai servizi *online* resi disponibili da Google: *Maps*, *Drive*, *Translate*, *YouTube*, ...
Per accedere a questi servizi è necessario registrare il progetto della *APP* per l'utilizzo di ogni singolo servizio.

1. La chiave rilasciata per lo sviluppo e il *debug* della *APP* dovrà essere successivamente sostituita con quella generata per la versione definitiva da rilasciare.

```
    android:supportsRtl="true"
    android:theme="@style/AppTheme">

    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>

    <service
      android:name=".LocationService"
      android:enabled="true"
      android:exported="false"/>

    <meta-data
      android:name="com.google.android.geo.API_KEY"
      android:value="..............................."/>

    <activity
      android:name=".MapsActivity"
      android:label="@string/title_activity_maps"
      android:parentActivityName=".MainActivity"/>

    <activity
      android:name=".InfoActivity"
      android:label="@string/title_activity_info"
      android:parentActivityName=".MainActivity"/>

    </application>
</manifest>
```

Classe *GeoTrackerActivity*

Per la *APP* è necessario prevedere un sistema per il controllo e la richiesta di permessi a tempo di esecuzione: per fare questo, le *activity* estendono la classe *GeoTrackerActivity* a sua volta derivata da *PermissionCompatActivity* definita nel capitolo B1. Questa classe, oltre a ereditare le funzionalità della superclasse, registra la *activity* attiva come *activity* corrente della *APP*, in modo da permettere al *service* per la ricezione degli aggiornamenti di posizione di effettuare la richiesta dei permessi necessari. Infine definisce il metodo *onServiceStatusChanged*, invocato dall'oggetto *application* a ogni cambiamento dello stato di esecuzione del *service*.

GeoTrackerActivity.java

```
import android.content.Intent;

public class GeoTrackerActivity extends PermissionCompatActivity {

  // notifica alla classe GeoTrackerApplication di essere
  // la activity correntemente attiva tra quelle della APP
  public void onStart() {

    super.onStart();

    GeoTrackerApplication app = (GeoTrackerApplication) getApplication();
```

```
    app.setCurrentActivity(this);
  }

  // metodo invocato quando il service che riceve gli aggiornamenti GPS
  // cambia il suo stato da attivo a inattivo e viceversa
  public void onServiceStatusChanged() {
  }

  public void onPermissionDenied(int requestCode) {
    super.onPermissionDenied(requestCode);
    // cancella un'eventuale esecuzione pendente
    LocationService.cancelPendingRequest();
  }

  // metodo invocato quando una richiesta di permesso termina con esito
  // positivo
  public void onPermissionGranted(int requestCode) {
    super.onPermissionGranted(requestCode);

    // riavvia il service se l'esecuzione precedente era stata terminata per
    // la mancanza dei permessi necessari
    if (LocationService.hasPendingRequest())
      startService(new Intent(this, LocationService.class));
  }
}
```

Classe *GeoTrackerApplication*

La classe *PermissionCompatActivity* permette di controllare all'interno delle varie *activity* la disponibilità dei permessi necessari. Ma il componente che riceve gli aggiornamenti relativi alla posizione del dispositivo è un *service* che deve essere in grado di controllare, prima dell'esecuzione di alcune operazioni, se la *APP* possiede o meno i permessi necessari. A questo scopo la classe *GeoTrackerApplication* mantiene un riferimento alla *activity* corrente e richiede a essa eventuali controlli sullo stato dei permessi della *APP*. Inoltre questa classe conserva le informazioni relative al server indicato dall'utente per l'invio dei dati raccolti, utilizzando la API per le *SharedPreferences*.

GeoTrackerApplication.java

```
import android.app.Application;
import android.content.SharedPreferences;
import android.preference.PreferenceManager;
import android.util.Patterns;
import java.util.regex.Matcher;

public class GeoTrackerApplication extends Application {
  private GeoTrackerActivity currentActivity;

  // indicatore utilizzato dalla MainActivity per mostrare una o l'altra
  // versione della finestra delle impostazioni
  private boolean isFirstLaunch;
  private SharedPreferences sharedPreferences;
```

```
// dati inseriti dall'utente all'interno della MainActivity
private String accessCode = "";
private String server = "";
public final static int serverPort = 12345;

public void onCreate() {
  super.onCreate();
  sharedPreferences = PreferenceManager.getDefaultSharedPreferences(this);
  server =
  sharedPreferences.getString(getString(R.string.pref_server_key), "");
  accessCode = sharedPreferences.getString(
              getString(R.string.pref_access_code_key), "");
  setFirstLaunch();
}

public boolean isFirstLaunch() {
  return isFirstLaunch;
}

// viene impostata l'apertura di una finestra diversa se non sono stati
// inseriti i parametri di accesso
private void setFirstLaunch() {
  isFirstLaunch = accessCode.length() == 0 || server.length() == 0;
}

public String getAccessCode() {
  return accessCode;
}

public void setAccessCode(String accessCode) {
  // limita la lunghezza della stringa a un massimo di 32 caratteri
  this.accessCode = accessCode.substring(0,
                (32 > accessCode.length()) ? accessCode.length() : 32);
  SharedPreferences.Editor editor = sharedPreferences.edit();
  editor.putString(getString(R.string.pref_access_code_key), accessCode);
  editor.apply();
  setFirstLaunch();
}

public String getServer() {
  return server;
}

public void setServer(String server) {
  this.server = server;
  SharedPreferences.Editor editor = sharedPreferences.edit();
  editor.putString(getString(R.string.pref_server_key), server);
  editor.apply();
  setFirstLaunch();
}

// indica se l'utente ha inserito le informazioni necessarie per l'invio
// dei dati di posizione al server
public boolean isValidHost() {
  if(accessCode.isEmpty() || server.isEmpty())
    return false;
  Matcher matcher = Patterns.IP_ADDRESS.matcher(server);
```

```
    if (!matcher.matches())
      return false; // indirizzo IP non corretto
    return true;
  }

  // imposta la activity fornita come argomento come corrente
  public void setCurrentActivity(GeoTrackerActivity currentActivity) {
    this.currentActivity = currentActivity;
  }

  // inoltra la richiesta ricevuta alla activity corrente
  public boolean hasLocationPermissions() {
    if (currentActivity == null)
      return false;
    return currentActivity.checkLocationPermission();
  }

  // inoltra il cambiamento di stato del service alla activity corrente
  public void onServiceStatusChanged() {
    currentActivity.onServiceStatusChanged();
  }
}
```

Classi *LocationService* e *UDPThread*

Sono state entrambe introdotte nella possibile soluzione della simulazione di prova scritta.

Activity MainActivity

La *activity* principale della *APP* permette all'utente di iniziare la gara, di terminarla, di accedere alla mappa del percorso effettuato e alle informazioni relative alla gara in corso, o terminata; inoltre permette all'utente di inserire le informazioni relative al server a cui inviare i dati di posizione attraverso una finestra dedicata.

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:orientation="vertical"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context=".MainActivity">

  <Button
    android:id="@+id/start"
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:text="@string/start_journey"
  />
  <Button
    android:id="@+id/stop"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:enabled="false"
    android:text="@string/end_journey"
  />
  <Button
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/show_map"
  />
  <Button
    android:id="@+id/info"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/show_info"
  />
  <Button
    android:id="@+id/settings"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/show_settings"
  />
</LinearLayout>
```

MainActivity.java

```
import android.app.Dialog;
import android.os.Bundle;
import android.support.v7.app.AlertDialog;
import android.view.View;
```

```
import android.widget.*;
import android.content.*;

public class MainActivity extends GeoTrackerActivity
                          implements View.OnClickListener {
  private Button startJourney;
  private Button endJourney;
  private Button showMap;
  private Button showInfo;
  private Button showSetting;

  private AlertDialog error;
  private AlertDialog dataDialog;
  private AlertDialog.Builder dataDialogBuilder;
  private boolean isFirstAlert = true;
  private GeoTrackerApplication app;

  // listener per impostazione dati di accesso
  private DialogInterface.OnClickListener positiveListener =
    new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialogInt, int i) {
        if (app.isFirstLaunch())
          Toast.makeText(MainActivity.this, R.string.toast_first,
                         Toast.LENGTH_LONG).show();
        else
          Toast.makeText(MainActivity.this, R.string.toast_message,
                         Toast.LENGTH_LONG).show();
        Dialog dialog = (Dialog) dialogInt;
        EditText edit = (EditText) dialog.findViewById(R.id.access_code);
        app.setAccessCode(edit.getText().toString());
        edit = (EditText) dialog.findViewById(R.id.server);
        app.setServer(edit.getText().toString());
        changeDataDialog();
      }

    };

  // listener per cancellazione dati di accesso
  private DialogInterface.OnClickListener neutralListener =
    new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialogInt, int i) {
        app.setAccessCode("");
        app.setServer("");
        Toast.makeText(MainActivity.this, R.string.toast_erase,
                       Toast.LENGTH_LONG).show();
        changeDataDialog();
      }
    };

  // aggiorna lo stato dei pulsanti di inizio e fine
  private void updateUI() {
    if (LocationService.isRunning()) {
      startJourney.setEnabled(false);
      endJourney.setEnabled(true);
    } else {
      startJourney.setEnabled(true);
```

```
      endJourney.setEnabled(false);
    }
  }

  // imposta la finestra di pop-up per dati di accesso
  private void changeDataDialog() {
    if (isFirstAlert && !app.isFirstLaunch()) {
      dataDialogBuilder
        .setPositiveButton(R.string.btn_pos_default, positiveListener)
        .setNegativeButton(R.string.btn_neg_default, null)
        .setNeutralButton(R.string.btn_neutral, neutralListener);
    }
    if (!isFirstAlert && app.isFirstLaunch())
      dataDialogBuilder
        .setPositiveButton(R.string.btn_pos_first, positiveListener)
        .setNegativeButton(R.string.btn_neg_first, null)
        .setNeutralButton("", null);
    dataDialog = dataDialogBuilder.create();
    isFirstAlert = app.isFirstLaunch();
  }

  // costruisce la finestra di pop-up per i dati di accesso
  private void buildDataAlert() {
    dataDialogBuilder = new AlertDialog.Builder(this);
    dataDialogBuilder.setTitle(R.string.settings_title)
      .setCancelable(true)
      .setView(R.layout.form_layout)
      .setPositiveButton(R.string.btn_pos_first, positiveListener)
      .setNegativeButton(R.string.btn_neg_first, null)
      .setNeutralButton("", null);
    changeDataDialog();
    if (app.isFirstLaunch()) {
      showDataDialog();
    }
  }

  // mostra la finestra di pop-up per i dati di accesso
  private void showDataDialog() {
    dataDialog.show();
    TextView message = (TextView) dataDialog.findViewById(R.id.message);
    if (app.isFirstLaunch())
      message.setText(R.string.first_message);
    else
      message.setText(R.string.default_message);
    // imposta i valori dei componenti della finestra
    EditText edit = (EditText)dataDialog.findViewById(R.id.access_code);
    edit.setText(app.getAccessCode());
    edit = (EditText) dataDialog.findViewById(R.id.server);
    edit.setText(app.getServer());
  }

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

```
    startJourney = (Button) findViewById(R.id.start);
    startJourney.setOnClickListener(this);
    endJourney = (Button) findViewById(R.id.stop);
    endJourney.setOnClickListener(this);
    showMap = (Button) findViewById(R.id.map);
    showMap.setOnClickListener(this);
    showInfo = (Button) findViewById(R.id.info);
    showInfo.setOnClickListener(this);
    showSetting = (Button) findViewById(R.id.settings);
    showSetting.setOnClickListener(this);
    error = new AlertDialog.Builder(this)
      .setTitle(getString(R.string.alert_title))
      .setMessage(R.string.alert_message)
      .setCancelable(true)
      .setPositiveButton(R.string.alert_positive, null)
      .create();
    app = (GeoTrackerApplication) getApplication();
    updateUI();
    buildDataAlert();
  }

  public void onClick(View widget) {
    switch (widget.getId()) {
      case R.id.start:
        if (!LocationService.isRunning()) {
          startService(new Intent(this, LocationService.class));
        }
        break;
      case R.id.stop:
        if (LocationService.isRunning()) {
          stopService(new Intent(this, LocationService.class));
        }
        break;
      case R.id.map:
        Intent maplaunch = new Intent(MainActivity.this,
                                      MapsActivity.class);
        startActivity(maplaunch);
        break;
      case R.id.info:
        Intent infolaunch = new Intent(MainActivity.this, InfoActivity.class);
        startActivity(infolaunch);
        break;
      case R.id.settings:
        showDataDialog();
        break;
    }
  }

  // aggiorna l'interfaccia utente
  public void onServiceStatusChanged() {
    super.onServiceStatusChanged();
    updateUI();
  }
```

```
  // se l'utente non concede il permesso visualizza un messaggio
  public void onPermissionDenied(int requestCode) {
    super.onPermissionDenied(requestCode);
    error.show();
  }
}
```

Classe *ReverseGeocoder*

Questa classe implementa un servizio di *reverse geocoding* ed è una versione modificata di quella introdotta nel capitolo B1; invece di inviare direttamente una richiesta HTTP al *web service* REST, utilizza un oggetto di tipo *Geocoder* ed è questo a farsi carico di effettuare la richiesta al *web service*. L'interfaccia *OnReverseGeocodingListener* è invece la stessa e non è qui riportata.

ReverseGeocoder.java

```
import android.os.AsyncTask;
import android.content.Context;
import java.io.IOException;
import java.util.List;
import android.location.*

public class ReverseGeocoder {
  // metodo statico per avvio di una richiesta asincrona
  static void requestAddressFromLocation(int requestCode,
                                         Location location,
                                         OnReverseGeocodingListener listener,
                                         Context context) {
    ReverseGeocoderTask task = new ReverseGeocoderTask(context, requestCode,
                                                       listener);
    task.execute(location);
  }
}

class ReverseGeocoderTask extends AsyncTask<Location, Void, String> {
  private Context context;
  private int requestCode;
  private OnReverseGeocodingListener listener;
```

```
  public ReverseGeocoderTask(Context context, int requestCode,
                             OnReverseGeocodingListener listener) {
    super();
    this.context = context;
    this.requestCode = requestCode;
    this.listener = listener;
  }

  protected String doInBackground(Location... params) {
    Geocoder geocoder = new Geocoder(context);
    double latitude = params[0].getLatitude();
    double longitude = params[0].getLongitude();
    List<Address> addresses;
    String addressText = null;

    try {
      addresses = geocoder.getFromLocation(latitude, longitude, 1);
      if (addresses != null && addresses.size() > 0) {
        StringBuilder builder = new StringBuilder();
        Address address = addresses.get(0);
        for (int i = 0; i < address.getMaxAddressLineIndex(); i++) {
          builder.append(address.getAddressLine(i));
          if(i+1 < address.getMaxAddressLineIndex())
            builder.append(", ");
        }
        addressText = builder.toString();
        // in caso di errore il risultato è nullo
        if(addressText.length() < 1)
          addressText = null;
      }
    }
    catch (IOException exception) {
    }
    return addressText;
  }

  protected void onPostExecute(String result) {
    listener.onReverseGeocodingResult(requestCode, result);
  }
}
```

Activity MapsActivity

MapsActivity è la *activity* che visualizza una mappa con sovraimpresso il percorso effettuato durante la gara in corso, o conclusa. I dati del percorso sono caricati dal file «savedData.txt» creato dal service *LocationService*.

Il *layout* di questa *activity* impiega un componente dell'interfaccia utente che permette la visualizzazione della mappa denominato *SupportMapFragment* definito nelle API *Google Play Services*: la classe *SupportMapFragment* disegna sullo schermo una mappa ottenuta mediante la connessione alla rete Internet del dispositivo.

Il contenuto della *activity MapsActivity* è costituito interamente dal *fragment*.

activity_maps.xml

```
<fragment
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/map"
  android:name="com.google.android.gms.maps.SupportMapFragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context=".MapsActivity"
/>
```

OSSERVAZIONE L'attributo *class* del *tag fragment* specifica il tipo di *fragment* impiegato; nel caso specifico *SupportMapFragment* della libreria *Google Play Services*.

Definito il *layout*, per visualizzare il percorso è necessario leggere il contenuto del file contenente i valori di posizione e visualizzare sulla mappa il tracciato degli spostamenti dell'utente. Per ottenere questo risultato si usano alcune classi rese disponibili dalle API Google:

- *Marker*: rappresenta un punto su una mappa; ne viene visualizzato uno per ogni posizione registrata nel file, ma quelli compresi tra la prima e l'ultima posizione sono nascosti;
- *Polyline*: rappresenta una linea spezzata poligonale sulla mappa passante per tutti i *marker* a essa associati (può essere aperta o chiusa: per chiuderla è necessario aggiungere come ultimo il primo *marker*);
- *PolylineOptions*: classe che definisce le opzioni di visualizzazione sullo schermo di una *polyline*;
- *LatLng*: formato per la rappresentazione di una posizione geografica espressa come latitudine/longitudine e impiegato per rappresentare i punti in un sistema cartesiano (in questo caso i valori di latitudine/longitudine sono espressi come numeri interi);
- *GoogleMap*: rappresenta la mappa visualizzata sullo schermo e ne gestisce le proprietà (tipo di mappa, visualizzazione della posizione attuale, ...);
- *OnMapReadyCallback*: interfaccia che espone il metodo *onMapReady*, invocato quando la mappa è pronta per essere utilizzata;
- *OnMapClickListener*: *listener* che intercetta il tocco dell'utente sulla mappa.

Inoltre la *activity* effettua una richiesta asincrona di *reverse geocoding* per ottenere gli indirizzi di partenza e di arrivo da mostrare come testo dei relativi *marker*.

MapsActivity.java

```
import android.Manifest;
import android.graphics.Color;
import android.location.Location;
```

```
import android.os.Bundle;
import com.google.android.gms.maps.*;
import com.google.android.gms.maps.model.*;
import java.io.*;

// una classe che usa un fragment deve estendere FragmentActivity di cui
// AppCompatActivity e GeoTrackerActivity sono sottoclassi
public class MapsActivity extends GeoTrackerActivity
                          implements OnMapReadyCallback,
                                     GoogleMap.OnMapClickListener,
                                     OnReverseGeocodingListener {
  private static final int REQUEST_START_LOCATION_NAME = 0;
  private static final int REQUEST_END_LOCATION_NAME = 1;

  private GoogleMap map;
  private Marker firstMarker = null;
  private Marker lastMarker = null;
  private Location firstLocation = null;
  private Location lastLocation = null;
  private boolean isDrawn = false;
  // indica se il percorso è stato disegnato o meno

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_maps);
    // ottiene il fragment definito e avvia la creazione della mappa
    SupportMapFragment mapFragment =
                        (SupportMapFragment)getSupportFragmentManager()
                                            .findFragmentById(R.id.map);
    mapFragment.getMapAsync(this);
  }

  // metodo invocato una volta che la mappa è pronta per essere usata
  public void onMapReady(GoogleMap googleMap) {
  map = googleMap;
  // abilita la visualizzazione della posizione attuale
  if (this.checkPermission(Manifest.permission.ACCESS_FINE_LOCATION) &&
      this.checkPermission(Manifest.permission.ACCESS_COARSE_LOCATION))
    map.setMyLocationEnabled(true);
  // imposta il tipo di mappa ibrida
  map.setMapType(GoogleMap.MAP_TYPE_HYBRID);
  // abilita l'intercettazione dell'interazione utente/mappa
  map.setOnMapClickListener(this);
  if (!isDrawn)
    drawJourney();
    isDrawn = true;
  }

  // metodo invocato quando sono concessi i permessi di localizzazione
  public void onPermissionGranted(int requestCode) {
    super.onPermissionGranted(requestCode);
    try {
      map.setMyLocationEnabled(true);
    } catch (SecurityException exception) {
    }
  }
```

```
// disegna il percorso aggiornato sulla mappa se già disponibile,
// altrimenti verrà effettuato in seguito
public void onStart() {
  super.onStart();
  if (map != null) {
    drawJourney();
    isDrawn = true;
  }
}

// disegna il percorso sulla mappa
private void drawJourney() {
  // rimuove i disegni precedenti
  map.clear();
  firstMarker = null;
  lastMarker = null;
  firstLocation = null;
  lastLocation = null;

  BufferedReader reader;
  String line;
  // opzioni di visualizzazione del tracciato
  PolylineOptions polylineOptions = new PolylineOptions();
  polylineOptions.color(Color.BLUE);
  polylineOptions.width(20);
  // marcatore di posizione
  Marker marker = null;
  // posizione cartesiana sulla mappa
  LatLng latLng = null;
  String[] token;
  Location location = null;
  // lettura dei dati di posizione da file
  try {
    reader = new BufferedReader(new FileReader(getApplicationContext()
                                    .getFilesDir()+"/savedData.txt"));

    try {
      while ((line = reader.readLine()) != null) {
        token = line.split(",");
        location = new Location("");
        double lat = Double.parseDouble(token[1]);
        location.setLatitude(lat);
        double lon = Double.parseDouble(token[2]);
        location.setLongitude(lon);
        // conversione in formato LatLng
        latLng = new LatLng(location.getLatitude(),
                            location.getLongitude());
        MarkerOptions markerOptions;
        if (marker == null) {
          // primo marker
          markerOptions = new MarkerOptions()
                              .position(latLng)
                              .title("Partenza: " + latLng.toString())
                              .visible(true);
          firstMarker = marker = map.addMarker(markerOptions);
```

```
            firstLocation = location;
          } else {
            markerOptions = new MarkerOptions()
                                .position(latLng)
                                .title(latLng.toString())
                                .visible(false);
            marker = map.addMarker(markerOptions);
          }
          polylineOptions.add(marker.getPosition());
        }
        // l'ultimo marker è visibile
        if (marker != null) {
          lastMarker = marker;
          lastLocation = location;
          if (marker == firstMarker) {
            // l'ultimo marker coincide col primo
            marker.setTitle("Partenza/Arrivo: "+latLng.toString());
          }
          else {
            marker.setTitle("Arrivo: " + latLng.toString());
            marker.setVisible(true);
          }
          // impostazione della visualizzazione sull'ultima posizione
          map.moveCamera(CameraUpdateFactory.newLatLng(latLng));
          map.animateCamera(CameraUpdateFactory.zoomTo(15));
          // creazione del tracciato sulla mappa
          map.addPolyline(polylineOptions);
          // richieste asincrone di reverse geocoding
          ReverseGeocoder.requestAddressFromLocation(
                          REQUEST_START_LOCATION_NAME,
                          firstLocation, this, this);
          if (firstMarker != lastMarker) {
            ReverseGeocoder.requestAddressFromLocation(
                            REQUEST_END_LOCATION_NAME,
                            lastLocation, this, this);
          }
        }
      }
      finally {
        reader.close();
      }
    }
    catch (IOException exception) {
    }
  }

  // centratura della mappa su tocco dell'utente
  public void onMapClick(LatLng point) {
    map.animateCamera(CameraUpdateFactory.newLatLng(point));
  }

  // metodo invocato quando la richiesta asincrona di reverse geocoding
  // è terminata
  public void onReverseGeocodingResult(int requestCode, String result) {
```

```
    // il fallimento della richiesta non richiede alcuna operazione aggiuntiva
    if (result == null)
      return;
    switch (requestCode) {
      case REQUEST_START_LOCATION_NAME:
        String prefix = (firstMarker != lastMarker) ?
          "Partenza: " : "Partenza/Arrivo: ";
        firstMarker.setTitle(prefix + result);
        break;
      case REQUEST_END_LOCATION_NAME:
        lastMarker.setTitle("Arrivo: " + result);
        break;

    }
  }
}
```

Activity InfoActivity

InfoActivity è la *activity* che visualizza le informazioni di riepilogo della gara in corso, o terminata.

activity_info.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  android:shrinkColumns="1"
  tools:context=".InfoActivity">
```

```
<TableRow
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingTop="@dimen/activity_vertical_margin">
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/starting_point"
    android:textSize="@dimen/medium_text"
    />
  <TextView
    android:id="@+id/start_pos_name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text=""
    android:textSize="@dimen/medium_text"
    android:maxLines="5"
    />
</TableRow>
<TableRow
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingTop="@dimen/activity_vertical_margin">
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/ending_point"
    android:textSize="@dimen/medium_text"
    />
  <TextView
    android:id="@+id/end_pos_name"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text=""
    android:textSize="@dimen/medium_text"
    android:maxLines="5"
    />
</TableRow>
<TableRow
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingTop="@dimen/activity_vertical_margin">
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/distance"
    android:textSize="@dimen/medium_text"
    />
```

```
      <TextView
        android:id="@+id/distance"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text=""
        android:textSize="@dimen/medium_text"
        />
    </TableRow>
    <TableRow
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:paddingBottom="@dimen/activity_vertical_margin"
      android:paddingTop="@dimen/activity_vertical_margin">
      <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/elapsed_time"
        android:textSize="@dimen/medium_text"
        />
      <TextView
        android:id="@+id/time"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text=""
        android:textSize="@dimen/medium_text"
        />
    </TableRow>
    <TableRow
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:paddingBottom="@dimen/activity_vertical_margin"
      android:paddingTop="@dimen/activity_vertical_margin">
      <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/mean_speed"
        android:textSize="@dimen/medium_text"
        />
      <TextView
        android:id="@+id/speed"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text=""
        android:textSize="@dimen/medium_text"
        />
    </TableRow>
  </TableLayout>
```

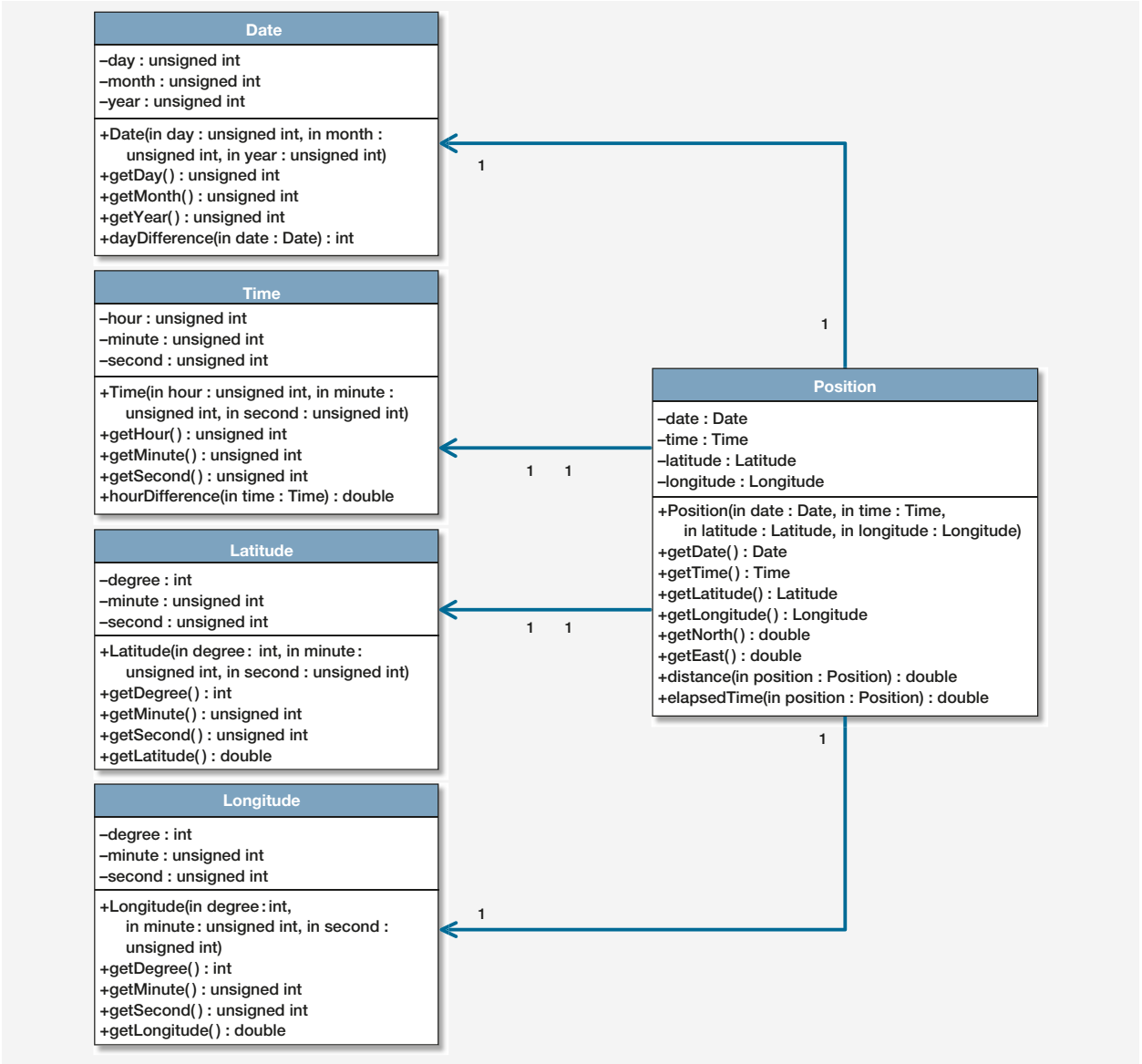
Le informazioni statistiche sono calcolate a partire dai dati salvati nel file testuale in formato CSV «savedData.txt» del quale ogni riga ha una struttura simile alla seguente:

```
3:22:25-9/12/2016,43.53900789,10.32166621,11.0,0.0827627
```

La stringa è suddivisa dai simboli «,» che delimitano i seguenti campi di dati:

- *timestamp*: orario e data UTC (*Universal Time Coordinated*) nel formato ora:minuti:secondi-giorno/mese/anno;
- latitudine: gradi decimali, positivo se il valore è relativo all'emisfero Nord, negativo altrimenti;
- longitudine: gradi decimali; positivo se indica un valore a Est di Greenwich, negativo altrimenti;
- altitudine: metri sul livello del mare (opzionale);
- velocità: metri al secondo (opzionale).

Per il calcolo dei valori da visualizzare sono state utilizzate classi per il calcolo accurato di distanze spaziali e temporali rappresentate dal diagramma UML seguente e contenute nel *package position*:



La classe *InfoActivity* implementa l'interfaccia *OnReverseGeocodingListener* per mostrare, se disponibili, gli indirizzi di partenza e di arrivo.

InfoActivity.java

```
import android.location.Location;
import android.os.Bundle;
import android.widget.TextView;
import it.zanichelli.android.geotracker.position.*;
import java.io.*;

public class InfoActivity extends GeoTrackerActivity
                          implements OnReverseGeocodingListener {
  private static final int REQUEST_START_LOCATION_NAME = 0;
  private static final int REQUEST_END_LOCATION_NAME = 1;
  private Position startPos = null;
  private Position endPos = null;
  private TextView startPosName;
  private TextView endPosName;

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_info);
    startPosName = (TextView) findViewById(R.id.start_pos_name);
    endPosName = (TextView) findViewById(R.id.end_pos_name);
  }

  // ogni volta che la activity viene mostrata, i dati visualizzati sono
  // aggiornati con i valori correnti
  public void onStart() {
    super.onStart();
    BufferedReader reader;
    String line;
    double totalDistance = 0.;
    int elapsedTime = 0;
    double meanSpeed = 0.;
    Position positionA = null;
    Position positionB;
    String[] token;
    String[] datetime;
    int hour, minute, second, day, month, year;
    Time time;
    Date date;
    Latitude lat;
    Longitude lon;

    try {
      reader = new BufferedReader(new FileReader(getApplicationContext()
                                  .getFilesDir()+"/savedData.txt"));
      try {
        while ((line = reader.readLine()) != null) {
        // lettura dati da file
          token = line.split(",");
          datetime = token[0].split(":|-|/");
          hour = Integer.parseInt(datetime[0]);
          minute = Integer.parseInt(datetime[1]);
          second = Integer.parseInt(datetime[2]);
          day = Integer.parseInt(datetime[3]);
          month = Integer.parseInt(datetime[4]);
```

```
        year = Integer.parseInt(datetime[5]);
        time = new Time(hour, minute, second);
        date = new Date(day, month, year);
        lat = new Latitude(Double.parseDouble(token[1]));
        lon = new Longitude(Double.parseDouble(token[2]));
        positionB = new Position(date, time, lat, lon);
        if (positionA == null) {
          // prima posizione registrata
          startPos = positionB;
        }
        else {
          // incremento della distanza complessiva percorsa
          totalDistance += positionA.distance(positionB);
        }
        endPos = positionA = positionB;
      }
    }
    finally {
      reader.close();
    }
  }
  catch (IOException exception) {
  }
  // calcola se possibile e visualizza le informazioni statistiche sul percorso
  if (startPos != endPos) {
    elapsedTime = startPos.elapsedTime(endPos);
    meanSpeed = totalDistance / (double) elapsedTime;
  }
  TextView text = (TextView) findViewById(R.id.distance);
  totalDistance = (int) (totalDistance * 100) / 100.;
  text.setText(totalDistance + " m ");
  text = (TextView) findViewById(R.id.time);
  hour = elapsedTime / 3600;
  minute = (elapsedTime % 3600) / 60;
  second = (elapsedTime % 3600) % 60;
  text.setText(intToString(hour)+":"+intToString(minute)+":"
               +intToString(second));
  text = (TextView) findViewById(R.id.speed);
  // conversione da m/s a km/h
  meanSpeed = (int) (meanSpeed * 360) / 100.;
  text.setText(meanSpeed + " km/h ");
  if (startPos != null) {
    Location startLocation = new Location("");
    startLocation.setLatitude(startPos.getLatitude().getLatitude());
    startLocation.setLongitude(startPos.getLongitude().getLongitude());
    startPosName.setText(startLocation.toString());
    // richiesta asincrona di reverse geocoding della/e posizione/i
    ReverseGeocoder.requestAddressFromLocation(REQUEST_START_LOCATION_NAME,
                                               startLocation, this, this);

    if (startPos != endPos) {
      Location endLocation = new Location("");
      endLocation.setLatitude(endPos.getLatitude().getLatitude());
      endLocation.setLongitude(endPos.getLongitude().getLongitude());
```

```
            endPosName.setText(endLocation.toString());
            ReverseGeocoder.requestAddressFromLocation(REQUEST_END_LOCATION_NAME,
                                                       endLocation, this, this);
          }
        }
      }

  // conversione da valore intero a stringa di due caratteri
  private String intToString(int val) {
    String string = Integer.toString(val);
    int len = string.length();

    if (len < 2)
      string = "0" + string;
      return s;
  }

// metodo invocato quando la richiesta asincrona di reverse geocoding
// è terminata
  public void onReverseGeocodingResult(int requestCode, String result) {
    if (result == null)
      return; // nessuna operazione da eseguire
    switch (requestCode) {
      case REQUEST_START_LOCATION_NAME:
        startPosName.setText(result);
        if (startPos == endPos)
          endPosName.setText(result);
        break;
      case REQUEST_END_LOCATION_NAME:
        endPosName.setText(result);
        break;
    }
  }
}
```