Lezione 23 – Geodecoding e SQLite

Geodecoding

Geocoding permette di traslare da un indirizzo a delle coordinate e viceversa. Questo permette di conoscera la posizione di un certo indirizzo o l'indirizzo piu' vicino di una certa posizione. La decodifica e' possibile interrogando un server remoto che contiene le posizione degli indirizzi. Per questa ragione l'applicazione che usa il geodecoding deve richiedere i permessi per accedere a Internet. Quindi nel manifesto dovra' comparire:

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

La claase Geodecoder fornisce due funzionalita': una per conoscere la posizioen di un indirizzo e l'altra data la posizione di trovare l'indirizzo. Tali operazioni vengono contestualizzate attraverso i locale in quanto gli indirizzi dipendono fortemente dal paese e lingua dell'utente.

```
Geocoder geocoder = new Geocoder(getApplicationContext(),
Locale.getDefault());
```

Tutti e due i mentodi ritornano una lista di oggetti Addres che rappresentano i possibili risultati. In fase di chimata si specifica il numero massimo di risultiati che si vuole vengano restituiti.
I dati contenuti nel singolo indirizzo sono tutti quelli che e' stato possibile recuperare nel database.

L'accesso al database viene effettuato in modo sincrono e quindi il thread si blocca finche' non riceve la risposta. Per questa ragione, le chiamate a questi metodi andrebbero fatte in thread secondari e non nel thread principale come negli esempi che seguono.

Reverse Geocoding: dalla posizione all'indirizzo

```
location =
locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
double latitude = location.getLatitude();
double longitude = location.getLongitude();
List<Address> addresses = null;

int maxResults=5;
Geocoder gc = new Geocoder(this, Locale.getDefault());
try {
addresses = gc.getFromLocation(latitude, longitude, maxResults);
} catch (IOException e) {}
```

L'accuratezza e la granularita' della risposta dipende dai dati presenti nel database remoto e che dipendono dalla localita' e dal tempo.

Forward Geocoding: dall'indirizzo alla posizione

Forward geocoding (o semplicemente geocoding) trova gli indirizzi corrispondenti a una stringa. La stringa contiene un nome di indirizzo. Ogni stringa con cui possiamo interrogare Google Maps e' adatta.

La lista di indirizzi restituita include la longitudine e la latidutine dell'indirizzo.

```
Geocoder fwdGeocoder = new Geocoder(this, Locale.US);
String streetAddress = "via Torino 155, Venezia";
List<Address> locations = null;

int maxResult=5;
try {
locations = fwdGeocoder.getFromLocationName(streetAddress, maxResult);
} catch (IOException e) {}
```

Per una localizzazione piu' precisa e' possibile usare il metodo overloaded con la possibilita' di specificare il rettangolo delimitatore della zona dove vogliamo cercare l'indirizzo, cosa che puo' risultare perticolarmente utile se vogliamo restringere la ricerca alla zona visualizzata all'utente (prossimo paragrafo).

```
List<Address> locations = null;
try {
locations = fwdGeocoder.getFromLocationName(streetAddress, 10,
n, e, s, w);
} catch (IOException e) {}
```

Esempio di utilizzo dell'oggetto Address:

```
private void updateWithNewLocation(Location location) {
String latLongString;
TextView myLocationText;
myLocationText = (TextView)findViewById(R.id.myLocationText);
String addressString = "No address found";
if (location != null) {
double lat = location.getLatitude();
double lng = location.getLongitude();
latLongString = "Lat:" + lat + "\nLong:" + lng;
double latitude = location.getLatitude();
double longitude = location.getLongitude();
Geocoder gc = new Geocoder(this, Locale.getDefault());
try {
List<Address> addresses = gc.getFromLocation(latitude, longitude, 1);
StringBuilder sb = new StringBuilder();
if (addresses.size() > 0) {
Address address = addresses.get(0);
for (int i = 0; i < address.getMaxAddressLineIndex(); i++)
sb.append(address.getAddressLine(i)).append("\n");
sb.append(address.getLocality()).append("\n");
sb.append(address.getPostalCode()).append("\n");
sb.append(address.getCountryName());
}
addressString = sb.toString();
} catch (IOException e) {}
} else {
latLongString = "No location found";
}
myLocationText.setText("Your Current Position is:\n" +
latLongString + "\n" + addressString);
}
```

SQLite

Anche Android prevede la possibilità di accedere ad un Database. L'API che viene usata non è però

JDBC. Anche il DB non è un DB server ma è implementato tramite un libreria C che gira nello stesso processo dell'applicazione Android (molto simile al Derby nella versione Embedded).

La modalità di accesso al DB è molto simile a quella di accesso ai Content Provider che vedremo nel prossimo paragrafo.

Ecco un esempio per la creazione di un database con una tabella:

```
SQLiteDatabase db;
db = openOrCreateDatabase(“DB”, Context.MODE_PRIVATE, null);
db.execSQL("create table PROVA  ( id  integer primary key autoincrement, Nome text
not null);”);
```

Una query al DB viene fatta utilizzando il metodo **query** che restituisce un cursore (istanza oggetto Cursor). Gli argomenti del/i metodo/i query sono:

1. un booleano equivalente alla keyword SQL DISTINCT;
2. il nome della tabella;
3. un'array di stringhe con i nomi delle colonne che compaiono nel risultato della query (proiezione);
4. L'equivalente della clausola WHERE che seleziona le righe da restituire. Si possono includere i caratteri speciali '?' Per gestire i parametri analogamente ai PreparedStatement.
5. Un'array di stringhe che rimpiazzeranno nell'ordine i parametri della clausola WHERE rappresentati da '?'.
6. La clausola GROUP BY che definisce come le righe devono essere raggruppate.
7. La clausola HAVING che seleziona i gruppi di righe da restituire
8. Una stringa che specifica l'ordine
9. Una stringa opzionale per limitare il numero di righe ritornate.

Es

```
String[] result_columns = new String[] {“id”, “nome”, “cognome”};
Cursor allRows = db.query(true, “PROVA”, result_columns,null, null, null, null, null,
null);

String where = “cognome=?”;
String order = “nome”;
String[] parameters = new String[]{requiredValue};
Cursor res = db.query(“PROVA”, null, where,parameters, null, null, order);

while(res.moveToFirst()) {

String nome = res.getString(“nome”);

}

// Creiamo una riga di valori da inserire
ContentValues newValues = new ContentValues();
// assegnare un valore per ogni colonna
newValues.put(“nome”, newValue);
[ ... Ripetere per ogni colonna ... ]
// inserire la riga nella tabella del DB
db.insert(“PROVA”, null, newValues);
```

```
// Creiamo una riga per modificare una riga nel DB
ContentValues updatedValues = new ContentValues();
// assegnare un valore per ogni  colonna
updatedValues.put(“nome”, newValue);
[ ... Ripetere per ogni colonna ... ]
String where = “id=?";
String[] parameters = new String[]{“3”};
// Aggiornare la riga nel DB
db.update(“PROVA”, updatedValues, where, parameters);

db.delete(“PROVA”, “nome=?", nnew String[]{“alessandro”});
db.close();
```