



Using SQLite in Android Apps



Overview

- SQLite databases vs. standard relational DBs.
- Creating and upgrading databases in Android callbacks.
- Need for background processes.
- Storing data in a database.
- Retrieving data from a database.
- Using a RecyclerView, not a ListView.

The logo graphic for SQLite, featuring a stylized 'S' composed of overlapping yellow, red, and blue squares, with a black crosshair intersecting them.

SQLite

- SQLite was created by D. Richard Hipp in 2000.
- SQLite is an open-source system and is in public domain (there is no license).
- The main Website is
www.sqlite.org
- It implements most of the SQL-92 standard.
- For practical applications, it is more than adequate.
- SQLite is not a server-type database system.
- Database is stored in a single file and application programs access the DB by accessing the file.

The logo consists of three overlapping squares: a yellow one at the top left, a red one at the bottom left, and a blue one at the bottom right. A black crosshair is superimposed on these squares.

SQLite

- SQLite columns can be of type:
 - INTEGER
 - REAL
 - TEXT
 - BLOB (Binary Large Object)
- These are *preferred types*, as SQLite is dynamically typed.
- A column defined as of some type may hold values of other types.

The logo consists of a yellow square, a red square, and a blue square arranged in a 2x2 grid, with a black crosshair overlaid. The word "SQLite" is written in a blue, sans-serif font to the right of the graphic.

SQLite

- SQLite uses **type affinity**
- The defined column types are treated as just *recommended* types
- The responsibility is on the application to write and then read in a semantically coherent manner
- However, in practical terms, it is a "normal" relational database system.
- It is the "in house" database in Android



Working with Databases

- Database schema (tables) must be defined first.
- This is normally done under Android's control by a special helper class.
- That class must extend Android's class `SQLiteOpenHelper` and override required methods.
- Usually, this class specifies how to create and upgrade the database.



Working with Databases

- Again, assume we are developing a simple app to store possible job leads, each including the company name, phone number, the company URL, and some comments we might want to save about the prospective employer.
- Instead of storing job leads in a file, as before, we will store them in an SQLite database.
- We start by designing a table to store our job leads (for a more complex app, we would likely need to create several tables).



Working with Databases

- In our app, we will use just one table to store job leads.
- Our table could be defined by the following SQL statement:

```
CREATE TABLE jobleads (  
  _id          INTEGER PRIMARY KEY AUTOINCREMENT,  
  name         TEXT,  
  phone        TEXT,  
  url          TEXT,  
  comments     TEXT  
)
```

AUTOINCREMENT
asks the DB to provide
unique values of the
primary key automatically

- We need to create a helper class to create and upgrade the database.



Working with Databases

- An app must always close a database that has been opened and no longer needed.
- Otherwise, memory leaks or other problems may occur.
- A simple solution is to make sure that there exists only one instance of `SQLiteOpenHelper` (i.e., a derived class), so that you can control db open/close operations easier.
- The **Singleton** design pattern is often used to this purpose.



Singleton Pattern Design Pattern

Name

- Singleton

Intent

- Ensure a class only has one instance, and provide a global point of access to it

Motivation

- It is important for some classes to have exactly one instance
- Make it illegal to have more than one instance, to be safe



Singleton Pattern Design Pattern

Motivation (cont.)

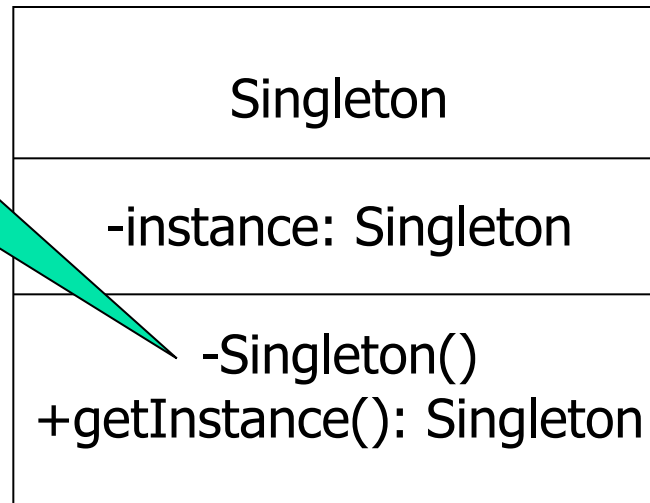
- Examples: there can be many printers in a system, but there should be only one printer spooler
- there should be only one object with a large state (internal data)
- creating lots of objects can take a lot of time
- extra objects take up memory
- it is a cumbersome to deal with different objects “floating” around if they are essentially the same



Singleton Pattern Design Pattern

Structure

private constructor!

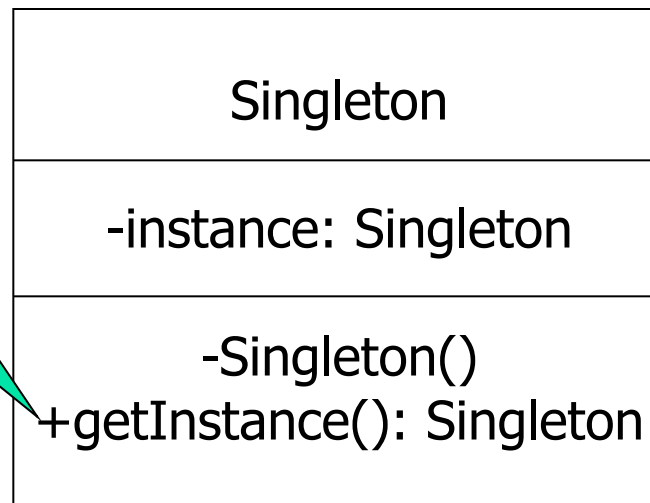




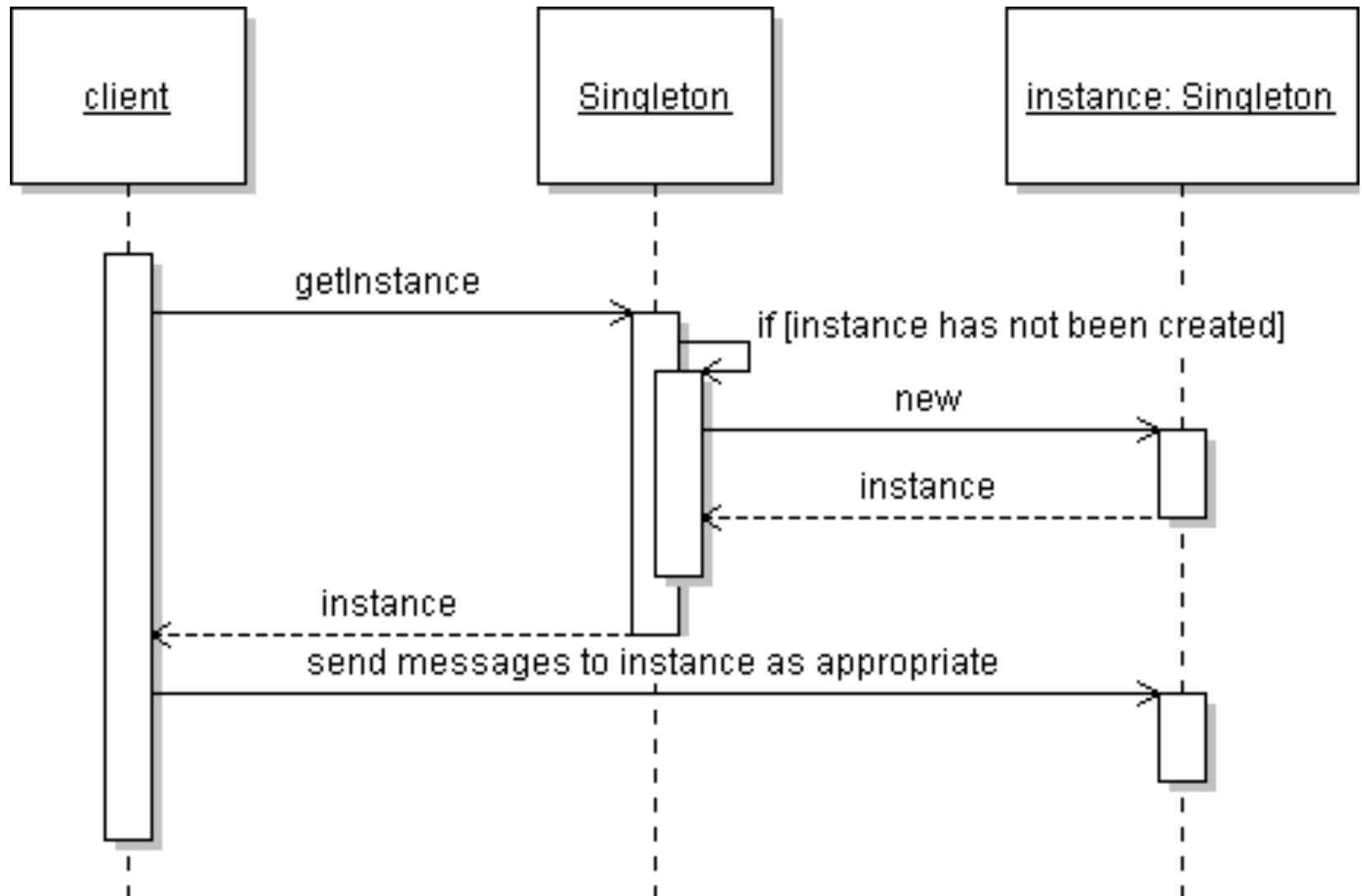
Singleton Pattern Design Pattern

Structure

the only access
point to the instance



Singleton Pattern Design Pattern





Singleton Pattern Design Pattern

Example code

```
public class Singleton {  
    private static Singleton instance = null;  
    private Singleton() {  
    }  
    public static synchronized Singleton getInstance() {  
        if(instance == null) {  
            instance = new Singleton();  
        }  
        return instance;  
    }  
}
```

A synchronized method can only be executed by one thread at a time

```
public class ExampleSingleton {  
    public static void main( String[] args ){  
        Singleton s = Singleton.getInstance();  
        ...  
    }  
}
```



Working with Databases

- Recap: our table is defined by the following SQL statement:

```
CREATE TABLE jobleads (  
  _id      INTEGER PRIMARY KEY AUTOINCREMENT,  
  name     TEXT,  
  phone    TEXT,  
  url      TEXT,  
  comments TEXT  
)
```

- We will now create a helper class to create and upgrade the database.



Extending SQLiteOpenHelper

```
public class JobLeadsDBHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "jobleads.db";
    private static final int    DB_VERSION = 1;
    private static JobLeadsDBHelper helperInstance;
    ...
    private JobLeadsDBHelper( Context context ) {
        super( context, DB_NAME, null, DB_VERSION );
    }
    public static synchronized JobLeadsDBHelper getInstance( Context context ) {
        if( helperInstance == null ) {
            helperInstance = new JobLeadsDBHelper( context.getApplicationContext() );
        }
        return helperInstance;
    }
    @Override
    public void onCreate( SQLiteDatabase db ) {
        ...
    }
    @Override
    public void onUpgrade( SQLiteDatabase db, int oldVersion, int newVersion ) {
        ...
    }
}
```

Private reference
to the single
instance

Private
constructor

public method
to access the
instance



Extending SQLiteOpenHelper

// it is convenient to have column names defined as Java constants

```
public static final String TABLE_JOBLEADS = "jobleads";  
public static final String JOBLEADS_COLUMN_ID = "_id";  
public static final String JOBLEADS_COLUMN_NAME = "name";  
public static final String JOBLEADS_COLUMN_PHONE = "phone";  
public static final String JOBLEADS_COLUMN_URL = "url";  
public static final String JOBLEADS_COLUMN_COMMENTS = "comments";
```

```
private static final String CREATE_JOBLEADS =  
    "create table " + TABLE_JOBLEADS + " ("  
        + JOBLEADS_COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "  
        + JOBLEADS_COLUMN_NAME + " TEXT, "  
        + JOBLEADS_COLUMN_PHONE + " TEXT, "  
        + JOBLEADS_COLUMN_URL + " TEXT, "  
        + JOBLEADS_COLUMN_COMMENTS + " TEXT"  
        + ")";
```

This method
will create the
database

```
@Override  
public void onCreate( SQLiteDatabase db ) {  
    db.execSQL( CREATE_JOBLEADS );  
}
```



Extending SQLiteOpenHelper

```
@Override
public void onUpgrade( SQLiteDatabase db, int oldVersion, int newVersion ) {
    db.execSQL( "drop table if exists " + TABLE_JOBLEADS );
    onCreate( db );
}
```

The above method will perform a database upgrade, if the old recorded version of the database (called DB_VERSION here and given to the parent class constructor) is lower than the new version.



Domain Objects

We will need a **domain class** (a POJO, or Plain Old Java Object) representing a job lead (like the previous one):

```
public class JobLead {  
    private long id;  
    private String companyName;  
    private String phone;  
    private String url;  
    private String comments;  
  
    public JobLead()  
    {  
        this.id = -1;  
        this.companyName = null;  
        this.phone = null;  
        this.url = null;  
        this.comments = null;  
    }  
    ...  
}
```



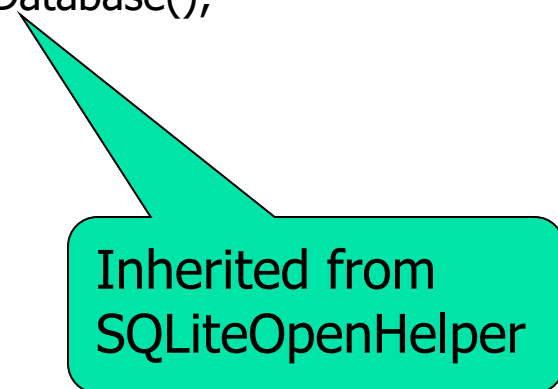
Working with Databases

- The JobLead class will be used to represent job leads within Java, while the `jobleads` table is used to represent (store) persistent object instances of the class in our database.
- We will transfer these objects back and forth between Java and our SQLite database, as needed.
- It is convenient to use another class to help transferring JobLead objects to and from our database.
- We will create a JobLeadsData class for this purpose.



Database Access Class

```
public class JobLeadsData {  
    private SQLiteDatabase db;  
    private SQLiteOpenHelper jobLeadsDbHelper;  
  
    public JobLeadsData( Context context ) {  
        this.jobLeadsDbHelper = JobLeadsDBHelper.getInstance( context );  
    }  
  
    public void open() {  
        db = jobLeadsDbHelper.getWritableDatabase();  
    }  
  
    public void close() {  
        if( jobLeadsDbHelper != null )  
            jobLeadsDbHelper.close();  
    }  
}
```



Inherited from
SQLiteOpenHelper

More information at:

developer.android.com/reference/android/database/sqlite/SQLiteDatabase



Database Access Class

```
public JobLead storeJobLead( JobLead jobLead ) {  
  
    ContentValues values = new ContentValues();  
  
    values.put( JobLeadsDBHelper.JOBLEADS_COLUMN_NAME, jobLead.getCompanyName());  
    values.put( JobLeadsDBHelper.JOBLEADS_COLUMN_PHONE, jobLead.getPhone() );  
    values.put( JobLeadsDBHelper.JOBLEADS_COLUMN_URL, jobLead.getUrl() );  
    values.put( JobLeadsDBHelper.JOBLEADS_COLUMN_COMMENTS, jobLead.getComments() );  
  
    // Insert the new row into the database table; the id (primary key) will be  
    // automatically generated by the database system and returned from db.insert  
    //  
    long id = db.insert( JobLeadsDBHelper.TABLE_JOBLEADS, null, values );  
  
    // store the id in the JobLead instance, as it is now persistent  
    jobLead.setId( id );  
  
    return jobLead;  
}
```

Storing the values
from the jobLead
argument object

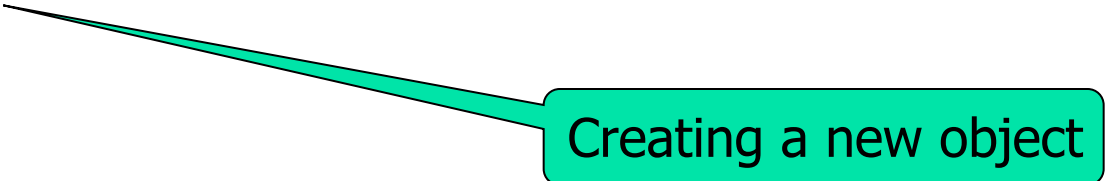
Inserting a new row



Database Access Class

```
public List<JobLead> retrieveAllJobLeads() {
    ArrayList<JobLead> jobLeads = new ArrayList<>();
    Cursor cursor = null;
    ...
    cursor = db.query( JobLeadsDBHelper.TABLE_JOBLEADS, allColumns,
                      null, null, null, null, null );
    while( cursor.moveToNext() ) {
        long id =
            cursor.getLong( cursor.getColumnIndex( JobLeadsDBHelper.JOBLEADS_COLUMN_ID ) );
        String name =
            cursor.getString( cursor.getColumnIndex( JobLeadsDBHelper.JOBLEADS_COLUMN_NAME ) );
        String phone =
            cursor.getString( cursor.getColumnIndex( JobLeadsDBHelper.JOBLEADS_COLUMN_PHONE ) );
        String uri =
            cursor.getString( cursor.getColumnIndex( JobLeadsDBHelper.JOBLEADS_COLUMN_URL ) );
        String comments = cursor.getString( cursor.getColumnIndex(
                                            JobLeadsDBHelper.JOBLEADS_COLUMN_COMMENTS ) );

        JobLead jobLead = new JobLead( name, phone, uri, comments );
        jobLead.setId( id );
        jobLeads.add( jobLead );
    }
}
```



Creating a new object



Using a Database Access Class

- We will use `JobLeadsData` in the Activities of our application to save and restore job leads.
- In the `onCreate` callback, we will create an instance of `JobLeadsData`
- It will have access to the *only object instance* of the `JobLeadsDBHelper` class (it is a *singleton!*).



Using a Database Access Class

- In the `onResume` callback, we should open the database.
- In the `onPause` callback, we should close the database, so that the DB will be closed if the activity happens to be killed by Android at this point.
- Depending on the type of the Activity (viewing our job leads or creating a new one), we will use the `retrieveAllJobLeads` **or** `storeJobLead` methods to interact with the database.



Using RecyclerView

- `RecyclerView` is a flexible alternative to a `ListView`, which is especially useful if a list of items is long.
- While a `ListView` is filled with all items at once, `RecyclerView` is filled only with a visible portion of the list (plus a few more items ahead and after, for buffering).
- List items that go out of the visible area, as the user scrolls through the list, are recycled and loaded with items that will become visible shortly.



Using RecyclerView

- Each item is controlled by a user-defined class which extends `RecyclerView.ViewHolder` class.
- Each element of the list has an associated layout which controls how the specific views of a list item are laid out in the user interface.
- For example, here is a simple layout, `job_lead.xml`, for displaying a job lead item.
- `CardView` is a useful View for displaying items in a list.



Using RecyclerView: an Item Layout

job_lead.xml layout:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
... >
    <androidx.cardview.widget.CardView
... >
        <LinearLayout
            <TextView
                android:id="@+id/companyName"
                ... " />
            <TextView
                android:id="@+id/phone"
                ... " />
            <TextView
                android:id="@+id/url"
                ... " />
            <TextView
                android:id="@+id/comments"
                ... " />
        </LinearLayout>
    </androidx.cardview.widget.CardView>
</LinearLayout>
```



Using RecyclerView: an Adapter

```
public class JobLeadRecyclerAdapter
    extends RecyclerView.Adapter<JobLeadRecyclerAdapter.JobLeadHolder> {

    private List<JobLead> jobLeadList;

    public JobLeadRecyclerAdapter( List<JobLead> jobLeadList ) {
        this.jobLeadList = jobLeadList;
    }

    class JobLeadHolder extends RecyclerView.ViewHolder {
        TextView companyName;
        TextView phone;
        TextView url;
        TextView comments;
        public JobLeadHolder( View itemView ) {
            super(itemView);
            companyName = (TextView) itemView.findViewById( R.id.companyName );
            phone = (TextView) itemView.findViewById( R.id.phone );
            url = (TextView) itemView.findViewById( R.id.url );
            comments = (TextView) itemView.findViewById( R.id.comments );
        }
    }
}
```



Using RecyclerView: an Adapter

@Override

```
public JobLeadHolder onCreateViewHolder( ViewGroup parent, int viewType ) {  
    View view = LayoutInflater.from( parent.getContext()).inflate( R.layout.job_lead, null );  
    return new JobLeadHolder( view );  
}
```

@Override

```
public void onBindViewHolder( JobLeadHolder holder, int position ) {  
    JobLead jobLead = jobLeadList.get( position );  
  
    holder.companyName.setText( jobLead.getCompanyName() );  
    holder.phone.setText( jobLead.getPhone() );  
    holder.url.setText( jobLead.getUrl() );  
    holder.comments.setText( jobLead.getComments() );  
}
```

@Override

```
public int getItemCount() {  
    return jobLeadList.size();  
}  
}
```



Asynchronous Method Calls

- However, reading and writing to our database may be a relatively slow process, which may impact the proper operation of the main UI thread.
- In other words, the dispatcher may be waiting for a handler (callback) return too long, and the device may appear to be **locked up**.
- In general, longer operations should be performed *asynchronously*, that is, in a separate thread (as *background* processes).



Asynchronous Method Calls

- An **asynchronous call** is a method call which **does not block the caller**. It just asks for the method to be executed.
- The called method executes, as usual, but the **caller method continues its own execution**, as well, without waiting for the method's result.
- The caller method and the called method **execute at the same time (concurrently)**, but in **different threads**, or in different processes.



Asynchronous Method Calls

- How does the called method provide the result of its computation back to the caller?
- This cannot work as a regular, *synchronous* call:

result = asyncMethod(arg);

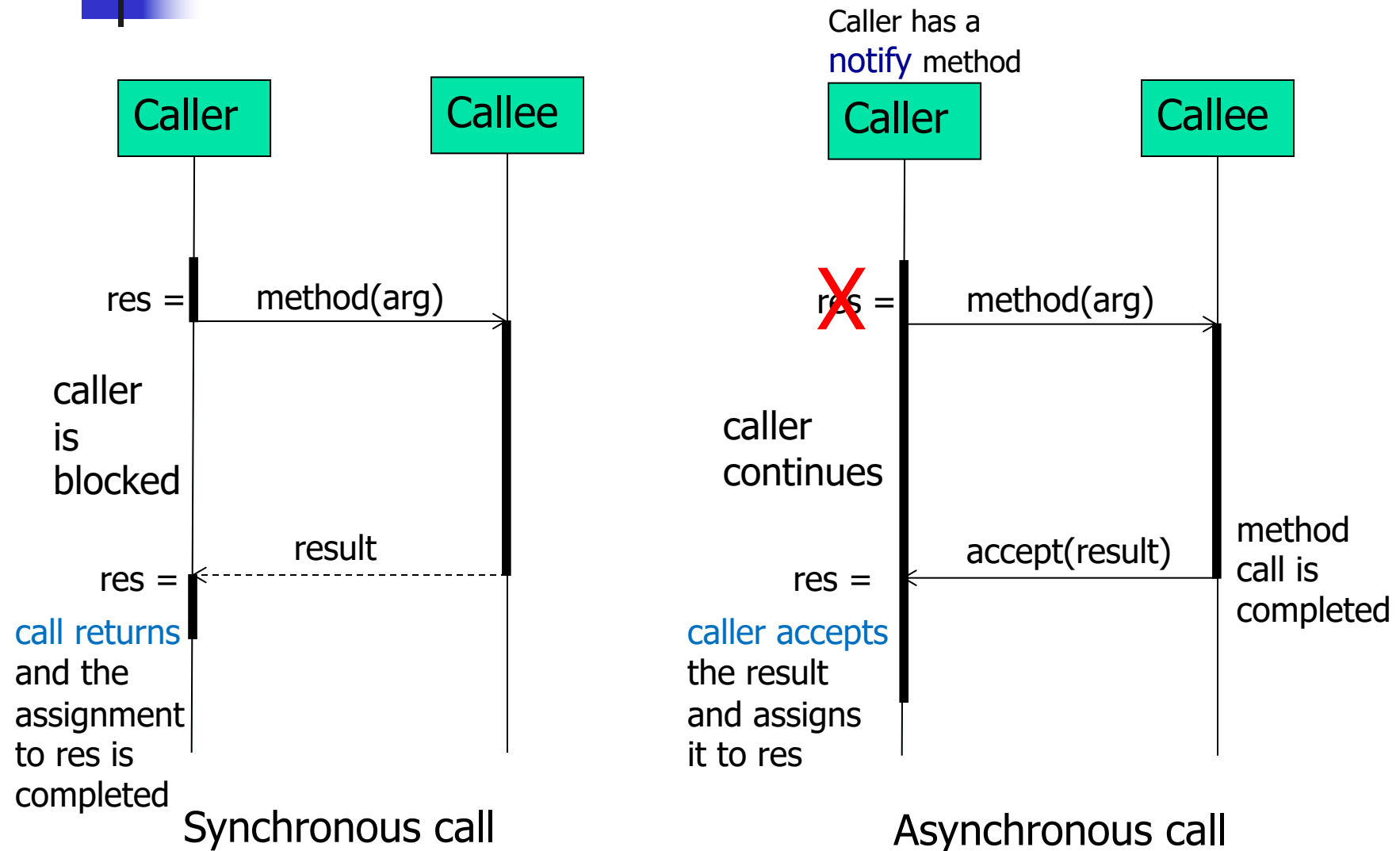
- Instead, the **caller needs to be notified**, when the method finished running and the result is ready.



Asynchronous Method Calls

- The caller provides a special method, say, `accept()`, for the called method to be invoked, once the called method finishes and want to return a result.
- The called method invokes the accept method upon completion.
- The argument to that method can be used to send the computed value back to the caller, as in `accept(result)`. **This method call executes in the caller's thread.**

Asynchronous Method Calls



accept() is really a callback!



Asynchronous Calls in Android

- In an Android app, we make such an asynchronous call from within some activity or fragment lifecycle callback method or a UI listener
- It can be either in an Activity or a Fragment, directly or indirectly (the callback may call some other method, which in turn calls the asynchronous method).



Asynchronous Calls in Android

- So, when an asynchronous method is called, the caller (the callback/listener method) simply completes its execution at some point and the control returns to the dispatcher thread, as usual; we are doing event-driven programming!
- So, the **UI thread can process other events**, while the background method executes. The Android device will not appear as “frozen”.



Asynchronous Calls in Android

- In Android, we can create such asynchronous processing by creating a class that *extends* the `AsyncTask` class and overrides a few important methods:

```
android.os.AsyncTask<Params, Progress, Result>
```

- **NOTE: this class has been deprecated in API 30 (Android 11).** While it still exists, nobody should use it anymore, but no official replacement exists. I implemented a replacement class for it, utilizing classes from the `java.util.concurrent` package.



Extending the AsyncTask Class

`android.os.AsyncTask<Params, Progress, Result>`

- `AsyncTask` is a generic class with three type parameters:
 - The first parameter, `Params`, is a type for the background process arguments (possibly many).
 - The second, `Progress`, is a unit type for indicating the progress of the background process.
 - The third, `Result`, is the result type of the background process.
 - `Void` can be used if no type will be used.



Extending the AsyncTask Class

`android.os.AsyncTask<Params, Progress, Result>`

- `AsyncTask` has 3 important methods:
 - `doInBackground(Params...)` to execute the background computation
 - `onPostExecute(Result)` to accept and process the result of the completed computation
 - `onProgressUpdate(Progress...)` to indicate the progress of the process when `publishProgress(Progress...)` is called from the background process.



Extending the AsyncTask Class

Java's syntax for variable number of parameters

`android.os.AsyncTask<Params, Progress, Result>`

- `AsyncTask` has 3 important methods:
 - `doInBackground(Params...)` to execute the background computation
 - `onPostExecute(Result)` to accept and process the result of the completed computation
 - `onProgressUpdate(Progress...)` to indicate the progress of the process when `publishProgress(Progress...)` is called from the background process.



Extending the AsyncTask Class

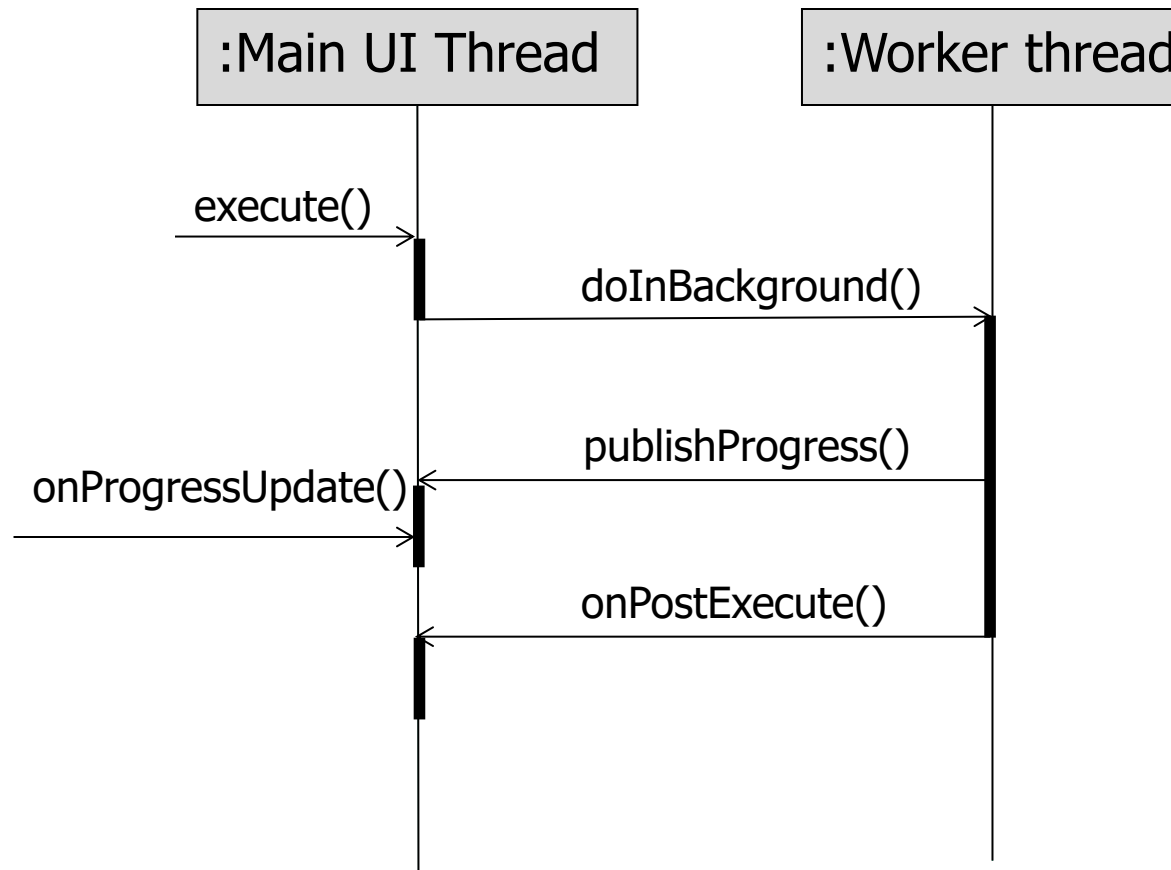
- A new, user-defined Task class should be created by extending `AsyncTask` and overriding *at least* `doInBackground` and `onPostExecute`.
- Other methods may be overridden, as needed.
- The asynchronous task is normally started in the main UI thread (one of the Activity/Fragment callbacks/listeners).



Extending the AsyncTask Class

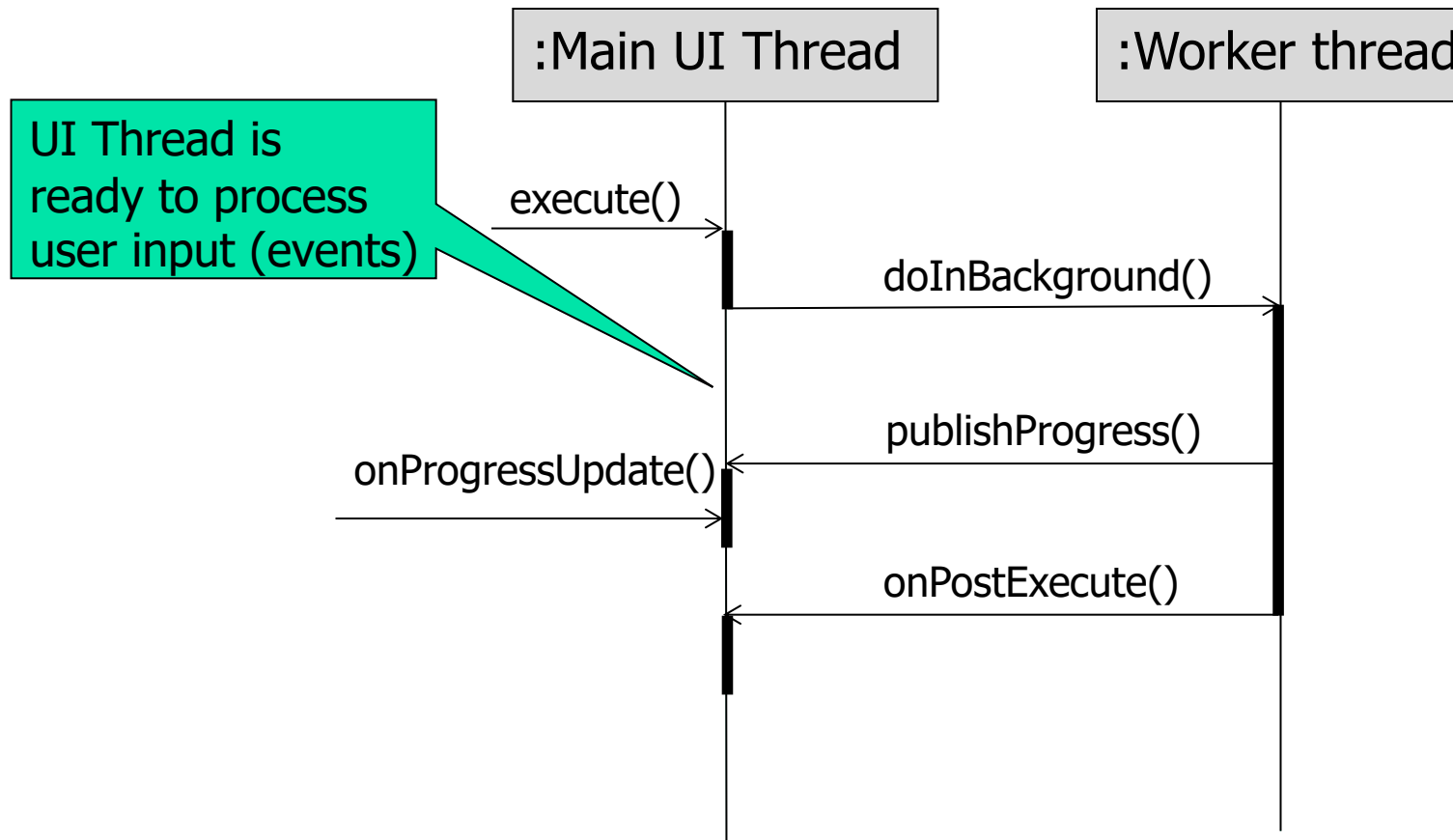
- In the main UI thread (e.g., in a button listener):
 - create an instance of the derived Task class (your subclass of `AsyncTask`), and
 - call the `execute(Params... params)` method on the instance, supplying the actual argument(s) for the background process.
- Android will then create a separate thread and call the overridden `doInBackground` in that separate thread.
- When it completes, Android will call `onPostExecute` *on the main UI thread*, supplying the result as argument.

Extending the AsyncTask Class



AsyncTask operation

Extending the AsyncTask Class



AsyncTask operation



@Override

@Override

```
Toast.makeText( getApplicationContext(), "Job lead created for " +
                jobLead.getCompanyName(),
                Toast.LENGTH_SHORT ).show();
```

}

}



Invoking Methods Asynchronously

A button listener code to save a new JobLead
asynchronously:

```
private class ButtonClickListener implements View.OnClickListener {
```

```
    @Override
```

```
    public void onClick( View v ) {
```

```
        String companyName = companyNameView.getText().toString();
```

```
        String phone        = phoneView.getText().toString();
```

```
        String url          = urlView.getText().toString();
```

```
        String comments     = commentsView.getText().toString();
```

```
        JobLead jobLead = new JobLead( companyName, phone, url, comments );
```

```
        // Execute the inserting of this new job lead in an asynchronous way,  
        // without blocking the UI thread.
```

```
        new JobLeadDBWriter().execute( jobLead );
```

```
    }
```

```
}
```




Another Example

```
public class JobLeadDBReader extends AsyncTask<Void, Void, List<JobLead>> {

    @Override
    protected List<JobLead> doInBackground( Void... params ) {
        jobLeadsData.open();
        jobLeadsList = jobLeadsData.retrieveAllJobLeads();
        return jobLeadsList;
    }

    @Override
    protected void onPostExecute( List<JobLead> jobLeadsList ) {
        super.onPostExecute(jobLeadsList);
        recyclerAdapter = new JobLeadRecyclerAdapter( jobLeadsList );
        recyclerView.setAdapter( recyclerAdapter );
    }
}
```

And to run it asynchronously in a callback or listener:

```
new JobLeadDBReader().execute();
```



Working with Assets and CSV Files

- Comma Separated Values (CSV) files are commonly used to represent tabular data.
- OpenCSV is one of the Java libraries for handling CSV files (many others exist).
- To include it in your app, edit build.gradle (Module: ...
and in the `dependencies` {... section, include:

`implementation 'com.opencsv:opencsv:5.9'`
- Then, synchronize the project:

File -> Sync Project with Gradle Files



Working with Assets and CSV Files

- A good location for a CSV file with some initial data for an app is the Assets folder. To create that folder:

File -> New -> Folder -> Assets Folder

Then accept the default placement and click Finish.

- You can simply copy/paste a CSV file into that folder.
- `OpenCSV` is a useful library to read and write CSV files.



Working with Assets and CSV Files

- A fragment of code to read from a CSV file called `data.csv` (a complete example app is on eLC):

```
InputStream ins = getAssets().open( "data.csv" );
```

```
CSVReader reader = new CSVReader( new InputStreamReader( ins ) );
```

```
String[] nextRow;
```

```
while( ( nextRow = reader.readNext() ) != null ) {
```

```
    for( int i = 0; i < nextRow.length; i++ ) {
```

```
        String nextCell = nextRow[i];
```

```
        ...
```



Summary

- We have learned how to work with SQLite databases in Android apps.
- We have learned how to perform database operations in a background process.
- We have learned how to store a new item in the database.
- We have learned how to retrieve all items from the database.
- We have learned how to display the retrieved items in a RecyclerView.



References and More Information

- Android Tools: "sqlite3":
 - <http://d.android.com/tools/help/sqlite3.html>
- SQLite:
 - <http://www.sqlite.org/>
- Command Line Shell For SQLite:
 - <http://www.sqlite.org/cli.html>
- Android API Guides: "Content Providers":
 - <http://d.android.com/guide/topics/providers/content-providers.html>
- Android SDK Reference regarding the application `android.database.sqlite` package:
 - <http://d.android.com/reference/android/database/sqlite/package-summary.html>
- Android SDK Reference regarding the application `AsyncTask` class:
 - <http://d.android.com/reference/android/os/AsyncTask.html>



References and More Information

- Android SDK Reference regarding the application ContentValues class:
 - <http://d.android.com/reference/android/content/ContentValues.html>
- Android SDK Reference regarding the application SQLiteDatabase class:
 - <http://d.android.com/reference/android/database/sqlite/SQLiteDatabase.html>
- Android SDK Reference regarding the application SQLiteOpenHelper class:
 - <http://d.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>
- Android SDK Reference regarding the application Cursor class:
 - <http://d.android.com/reference/android/database/Cursor.html>