Instituto Superior de Engenharia do Porto
(ISEP)

Licenciatura em Engenharia de Telecomunicações e Informática (LETI)

Desenvolvimento de Software e Sistemas Móveis (DSSMV)

DSSMV: PL: Week 4

ListView based App

Paulo Baltarejo Sousa and Carlos Filipe Freitas
{pbs,caf}@isep.ipp.pt
2025/26

1 List View

`ListView` is a view group that displays a list of scrollable items. The list items are automatically inserted to the list using an `Adapter` that pulls content from a source such as an array or database query and converts each item result into a view that's placed into the list.

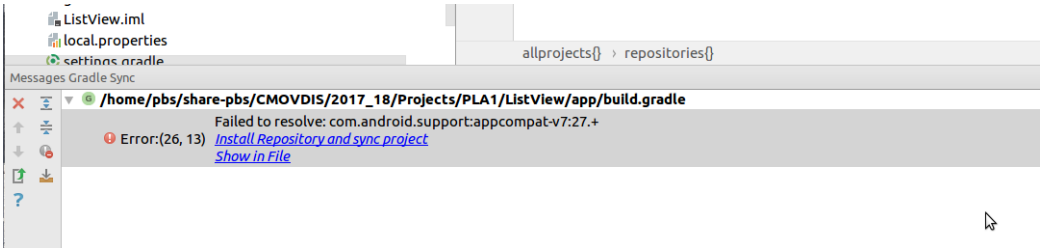
1.1 Create a project

Start a new Android project.

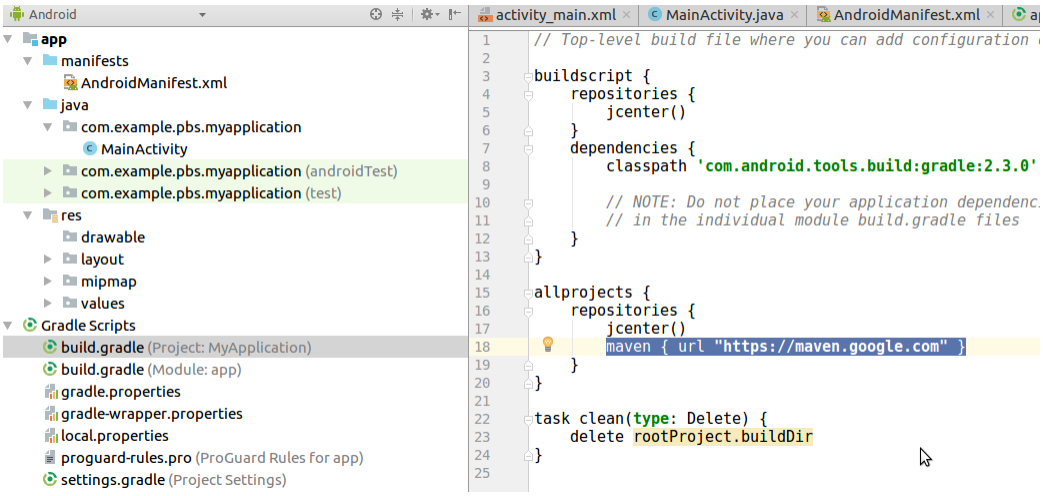
1. Select `File > New > New Project...`.
2. Fill in the project details with the following values:
   - New Project: Application name: List View
   - Target Android Devices: Phone and Tablet
   - Target Android Devices: Minimum SDK: API 16 (Jelly Bean)
   - Add an Activity to Mobile: Empty Activity
   - ...

1.2 Error

Pay attention: if you do not got this error, go to the next sub section. If you got this error message



add the following statement to the project gradle build file.
`maven url "https://maven.google.com"`



1.3 Add a `ListView` object to the Layout

Open the `activity_main.xml` file from the `res/layout/` directory.

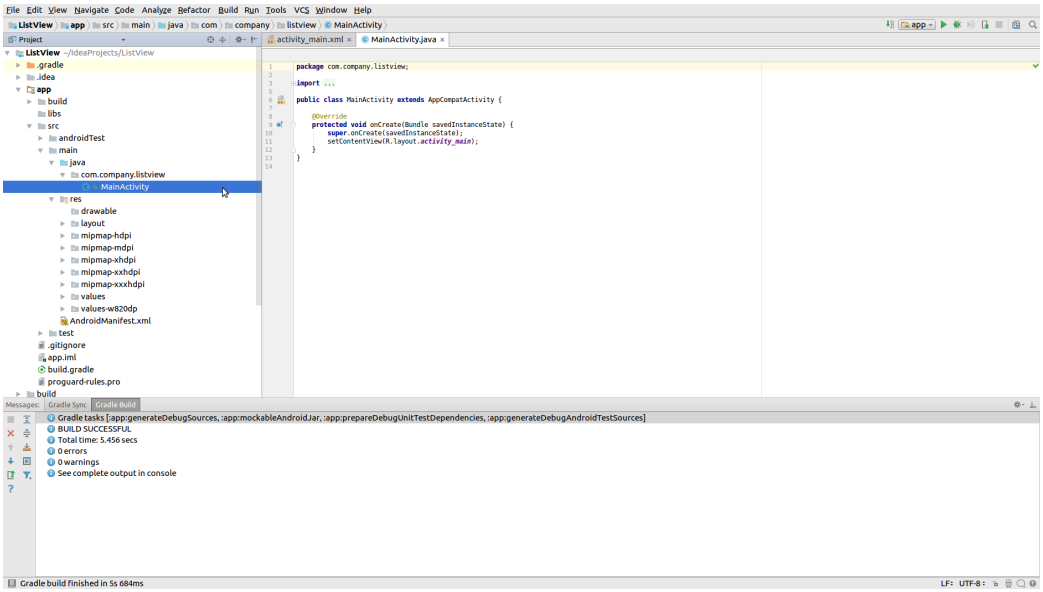
The `EmptyActivity` template you chose when you created this project includes the `activity_main.xml` file with a `android.support.constraint.ConstraintLayout` root view and a `TextView` child view.

First, replace that code by the following one:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:padding="10dp"
        tools:context=".MainActivity">

    <ListView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/listView"
            android:layout_alignParentTop="true"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true" />

</RelativeLayout>
```

1.4 Get a reference to the `ListView` object

From IntelliJ IDEA, open the `MainActivity.java` file, by double-clicking on `MainActivity`.



Get a reference to the `ListView` object included into the `activity_main.xml` layout file. For that, invoke `findViewById` to retrieve the reference to the `listView` element included in the `activity_main.xml` layout. The `ListView` object is identified by `listView`. Thus, it is possible to interact programmatically with such UI element.

```
public class MainActivity extends AppCompatActivity {

    private ListView lv;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        lv = (ListView) findViewById (R.id.listView);
    }
}
```

1.5 Get Data to fill `ListView` object

The input to the `ListView` object can be arbitrary Java objects. Thus, the source of data to fill the `ListView` object could be either a SQLite database or a file, just to mention some. In this case, it is used an XML resource. Therefore, add a string resource to `res/values/strings.xml` file:

```
<resources>
    <string name="app_name">List View</string>

    <string-array name="students">
        <item>Pedro Almeida</item>
        <item>Paulo Ribeiro</item>
        <item>Rita Sousa</item>
        <item>Matilde Sousa</item>
        <item>Matilde Coelho</item>
        <item>Dinis Sousa</item>
        <item>Carla Pereira</item>
        <item>Maria Vieira</item>
        <item>Joaquim Vieira</item>
        <item>Maria Santos</item>
        <item>Carlos Freitas</item>
        <item>Pedro Coelho</item>
    </string-array>

</resources>
```

In order to handle such data set programmatically, could be used an `ArrayList` object:

```
public class MainActivity extends AppCompatActivity {

    private ListView lv;
    private ArrayList<String> list;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        lv = (ListView) findViewById (R.id.listView);

        list = new ArrayList<String>();
        String[] students = getResources().getStringArray(R.array.students);
        Collections.addAll(list, students);
    }
}
```

1.6 Adapter

So far, there is the view and the data set. So, the next step is fill each row of the `ListView` object with items from data set. An `Adapter` object acts as a bridge between an `AdapterView` and the underlying data set for that view [1]. In this case, the `AdapterView` is each row of the `ListView` object and the data set is the `ArrayList` object.

An adapter manages the data set and adapts it to the individual rows in the `ListView` object. Every row in the `ListView` object has a layout, which could be an android built-in or customized layout.

```
public class MainActivity extends AppCompatActivity {

    private ListView lv;
    private ArrayList<String> list;
    private ArrayAdapter<String> adapter;

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

        lv = (ListView) findViewById (R.id.listView);

        list = new ArrayList<String>();
        String[] students = getResources().getStringArray(R.array.students);
        Collections.addAll(list, students);

        adapter = new ArrayAdapter<String>(/*context*/this,
                /*layout of each row*/android.R.layout.simple_list_item_1,
                /*data set*/list);

    }
}
```

Then the next step, is to assign the adapter to the `ListView` object. This is

[1] `http://developer.android.com/guide/topics/ui/declaring-layout.html#AdapterViews`

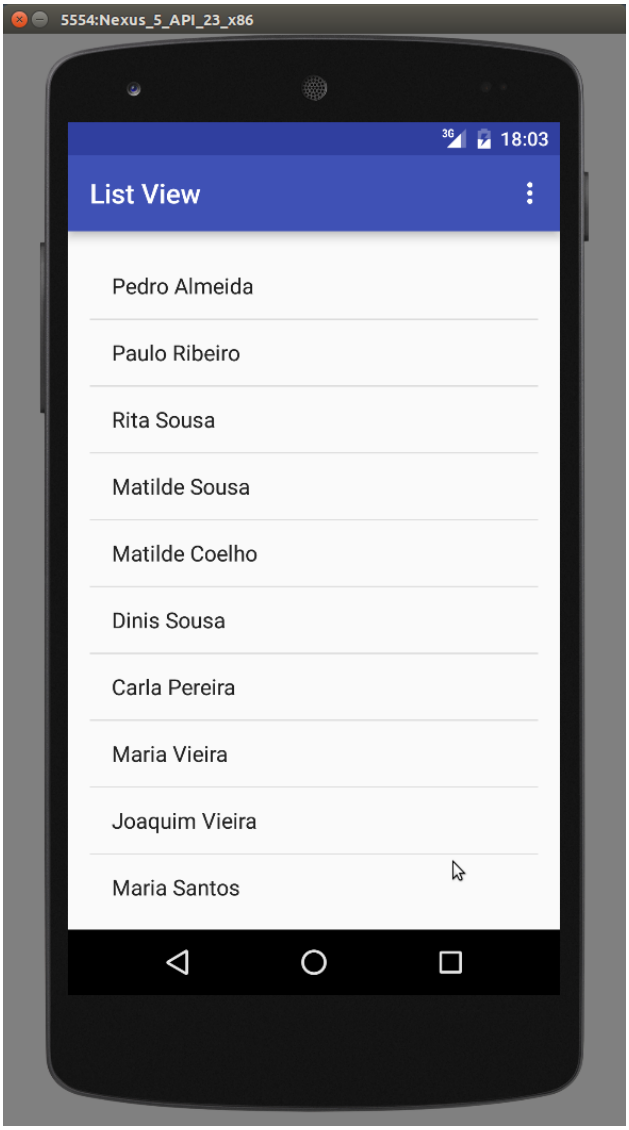
done via the `setAdapter` method on the `ListView` object.

```
public class MainActivity extends AppCompatActivity {

    private ListView lv;
    private ArrayList<String> list;
    private ArrayAdapter<String> adapter;

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

        lv = (ListView) findViewById (R.id.listView);

        list = new ArrayList<String>();
        String[] students = getResources().getStringArray(R.array.students);
        Collections.addAll(list, students);
        adapter = new ArrayAdapter<String>(/*context*/this,
                /*layout of each row*/android.R.layout.simple_list_item_1,
                /*data set*/list);
        lv.setAdapter(adapter);
    }
}
```

Using this approach, whenever the data set is changed the `ArrayAdapter` object must be notified by invoking its `notifyDataSetChanged` method.

1.7 Running on the Emulator

1. To run the app from IntelliJ IDEA: Click `Run` from the toolbar.



2 Menus

Menus are a common user interface component in many types of apps. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.

Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated Menu button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an action bar to present common user

actions.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs:

Options menu and action bar : The options menu is the primary collection of menu items for an activity. It is where you should place actions that have a global impact on the app, such as "Search", "Compose email", and "Settings."

On Android 2.3 and lower, users can reveal the options menu panel by pressing the Menu button. On Android 3.0 and higher, items from the options menu are presented by the action bar as a combination of on-screen action items and overflow options.

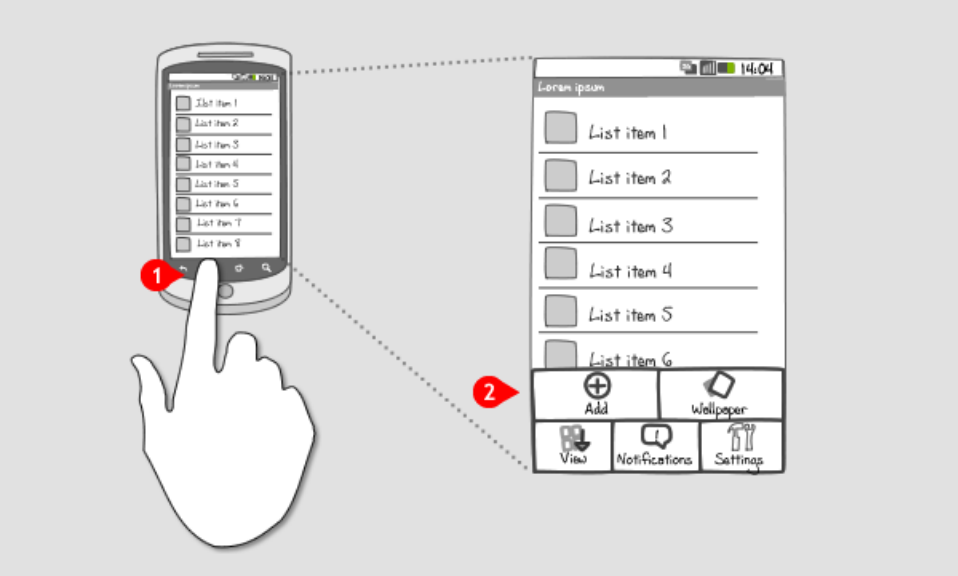
Context menu and contextual action mode : A context menu is a floating menu that appears when the user performs a long-press on an element. It provides actions that affect the selected content or context frame.

3 Creating an Option Menu

The option menu is where you should include actions and other options that are relevant to the current activity context, such as "Add", "Settings" and others generic actions.

Where the items in your options menu appear on the screen depends on the version for which you've developed your application:

- For Android 2.3.x (API level 10) or lower, the contents of the option menu appears at the bottom of the screen when the user presses the Menu button.



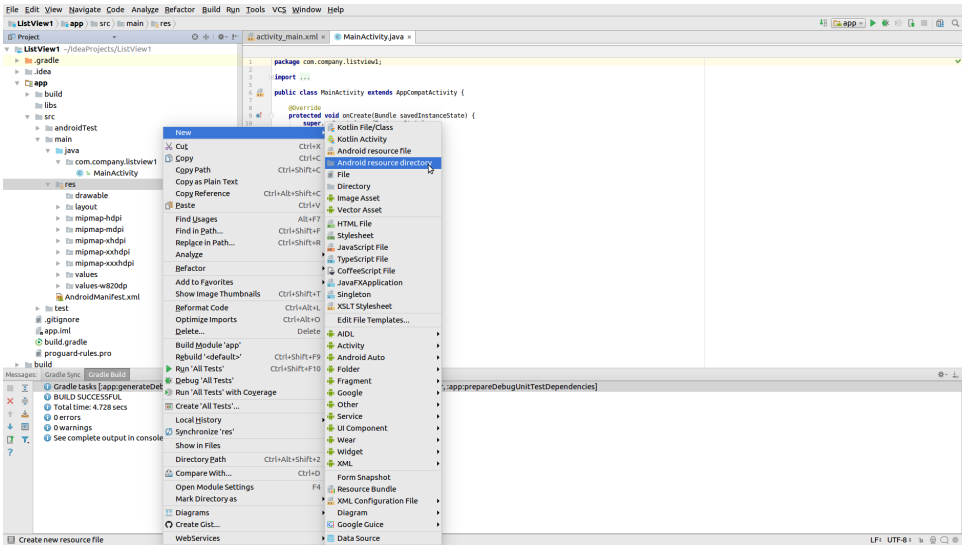
- For Android 3.0 (API level 11) and higher, items from the options menu are available in the action bar. By default, the system places all items in the action overflow, which the user can reveal with the action overflow icon on the right side of the action bar.

3.1 Creating a Menu Resource

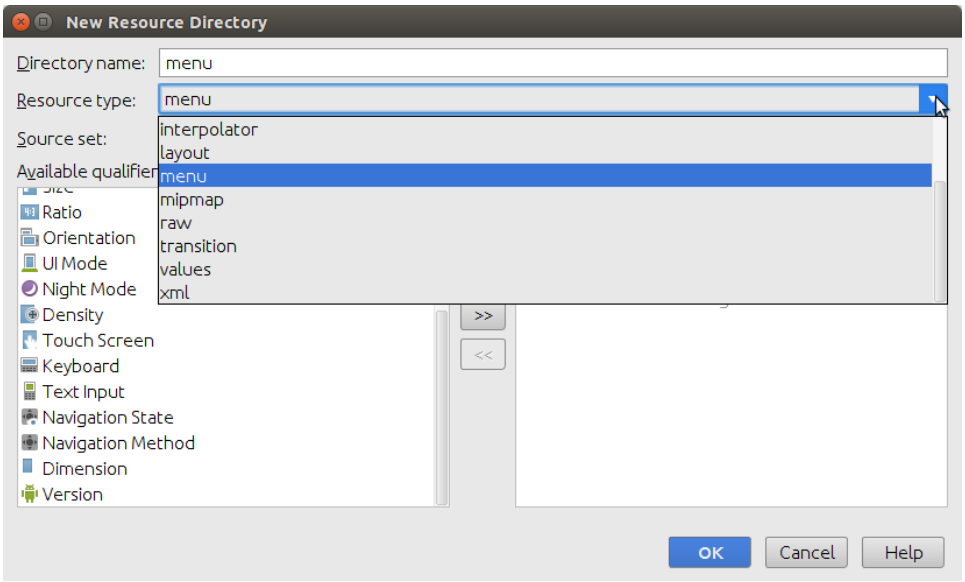
Instead of instantiating a `Menu` in your application code, you should define a menu and all its items in an XML menu resource, then inflate the menu resource (load it as a programmable object) in your application code. Using a menu resource to define your menu is a good practice because it separates the content for the menu from your application code. It's also easier to visualize the structure and content of a menu in XML.

Create a new resource directory in the `res/` directory.

1. Right click on `res/` and `New > Android resource directory`

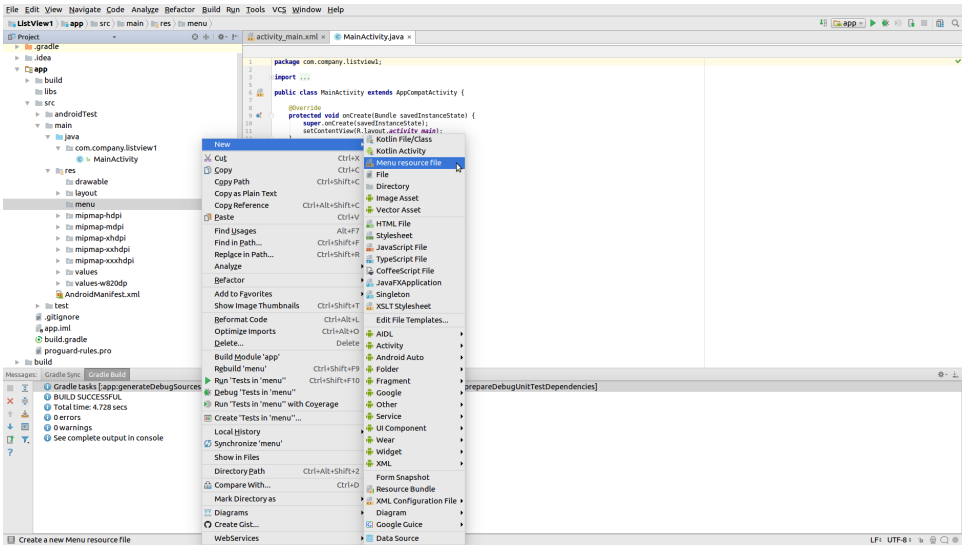


2. Choose `Resource Type` menu and click `Ok`.

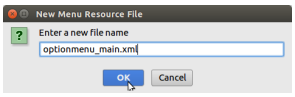


To create a menu resource, create an XML file inside your project's `res/menu/` directory:

1. Right click on `res/menu/` and choose `New > Menu resource file`



2. Enter the file name `optionmenu_main.xml` and click `Ok`.



To create a menu, you can use the following elements:

`menu` : Defines a Menu, which is a container for menu items. A `menu` element must be the root node for the file and can hold one or more `item` and `group` elements.

`item` : Creates a `MenuItem`, which represents a single item in a menu. This element may contain a nested `menu` element in order to create a sub-menu.

`group` : An optional, invisible container for `item` elements. It allows you to categorize menu items so they share properties such as active state and visibility.

Change the original content of `res/menu/optionmenu_main.xml` file to:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
           android:id="@+id/add"
           android:title="Add"/>
    <item
           android:id="@+id/fav"
           android:title="Favourite"/>
    <item
           android:id="@+id/settings"
           android:title="Settings"/>
</menu>
```

This example defines a menu with three items. Each item includes the attributes:

`android:id` : A resource ID that's unique to the item, which allows the application can recognize the item when the user selects it.

`android:title` : A reference to a string to use as the item's title.

3.2 Inflating a Menu Resource

From your application code, you can inflate a menu resource (convert the XML resource into a programmable object) using `MenuInflater.inflate`. For example, the following code inflates the `optionmenu_main.xml` file defined above, during the `onCreateOptionsMenu` callback method, to use the menu as the activity's Options Menu:

```
public class MainActivity extends AppCompatActivity {

    private ListView lv;
    private ArrayList<String> list;
    private ArrayAdapter<String> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ...
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater menuInflater = (MenuInflater) getMenuInflater();
        menuInflater.inflate(R.menu.optionmenu_main,menu);
        return true;
    }
}
```

The `getMenuInflater` method returns a `MenuInflater` for the activity. With this object, you can call `inflate` method, which inflates a menu resource into a Menu object. In this example, the menu resource defined by `optionmenu_main.xml` file is inflated into the Menu that was passed into `onCreateOptionsMenu`.

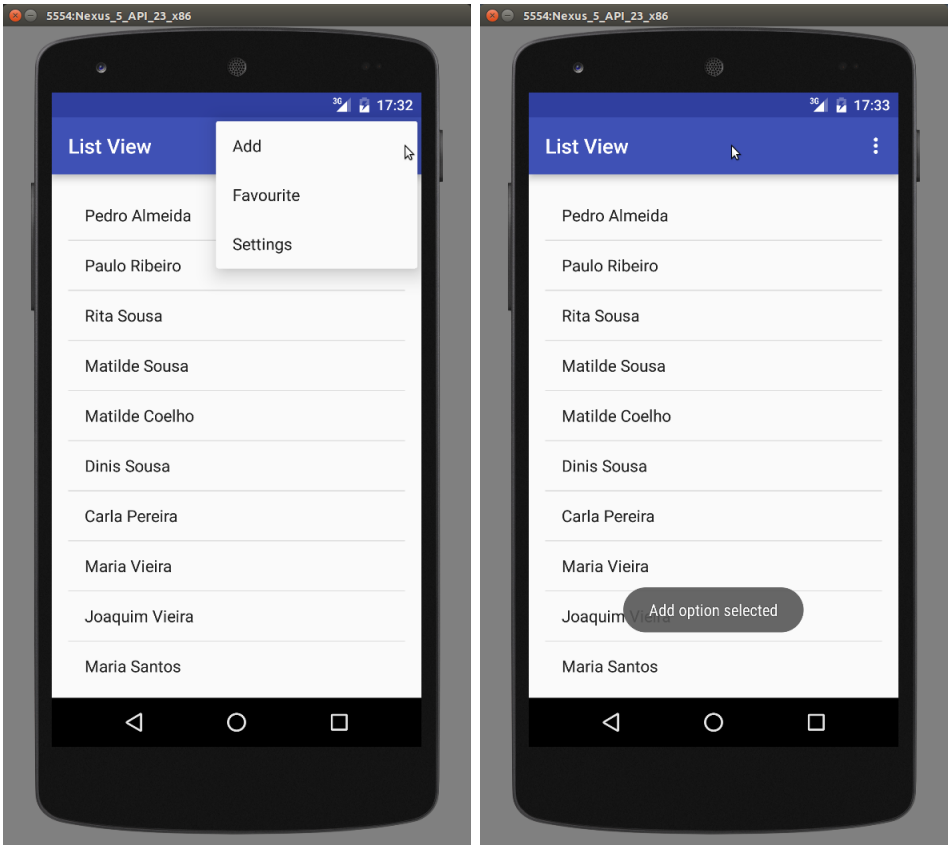
3.3 Handling click events

When the user selects an item from the option menu, the system calls your activity's `onOptionsItemSelected` method. This method passes the `MenuItem` selected. You can identify the item by calling `getItemId`, which returns the unique ID for the menu item (defined by the `android:id` attribute in the menu resource). You can match this ID against known menu items to perform the appropriate action. For example:

```
public class MainActivity extends AppCompatActivity {

    private ListView lv;
    private ArrayList<String> list;
    private ArrayAdapter<String> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
       ...
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
      ...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.add:
                Toast.makeText(getApplicationContext(),
                       "Add option selected", Toast.LENGTH_LONG).show();
                return true;
            case R.id.fav:
                Toast.makeText(getApplicationContext(),
                       "Favourite option selected", Toast.LENGTH_LONG).show();
                return true;
            case R.id.settings:
                Toast.makeText(getApplicationContext(),
                       "Settings option selected", Toast.LENGTH_LONG).show();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

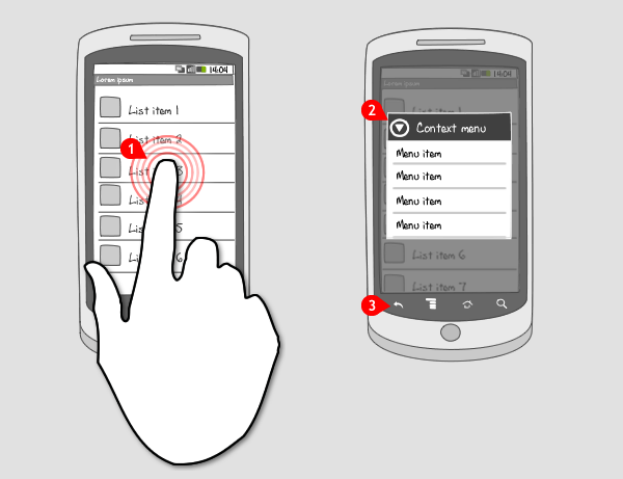
3.4 Running

1. To run the app from IntelliJ IDEA: Click `Run` from the toolbar.



4 Context Menu

A contextual menu offers actions that affect a specific item or context frame in the UI. You can provide a context menu for any view, but they are most often used for items in a `ListView`, `GridView`, or other view collections in which the user can perform direct actions on each item. A context menu is displayed when the user performs a "long press" (press and hold) on an item.

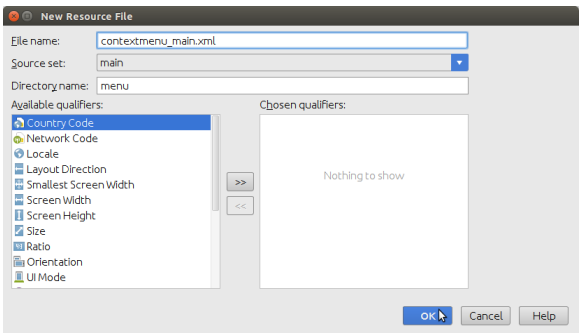


4.1 Creating a Context menu

Like option menus, context menus must be specified in an XML file. So the first step is to create a new XML file in the `res/menu` directory.

1. Rigth-click on the `res/menu` directory and select `New > Menu resource file`.

2. Name it and click `OK`



3. Add items to the menu. Here is an example of context menu (`res/menu/contextmenu_main.xml` file).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
           android:id="@+id/delete"
           android:title="Delete"/>
    <item
           android:id="@+id/edit"
           android:title="Edit"/>

</menu>
```

4.2 Inflating a Menu Resource

From your application code, you can inflate a menu resource (convert the XML resource into a programmable object) using `MenuInflater.inflate`. For example, the following code inflates the `contextmenu_main.xml` file defined above, during the execution of `onCreateContextMenu` method:

```
public class MainActivity extends AppCompatActivity {

    private ListView lv;
    private ArrayList<String> list;
    private ArrayAdapter<String> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        ...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        ...
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        MenuInflater menuInflater = (MenuInflater) getMenuInflater();
        menuInflater.inflate(R.menu.contextmenu_main,menu);
    }

}
```

4.3 Register for Context Menu

In order a `View` provide a context menu, you must "register" the view for a context menu. Call `registerForContextMenu` method and pass it to the `View` you want to give a context menu. When this `View` receives a long-press,

it displays a context menu. In this case, the `View` is the `ListView` object called `lv`.

```
public class MainActivity extends AppCompatActivity {

    private ListView lv;
    private ArrayList<String> list;
    private ArrayAdapter<String> adapter;

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

        lv = (ListView) findViewById (R.id.listView);

        list = new ArrayList<String>();
        String[] students = getResources().getStringArray(R.array.students);
        Collections.addAll(list, students);

        adapter = new ArrayAdapter<String>(/*context*/this,
               /*layout of each row*/android.R.layout.simple_list_item_1,
               /*data set*/list);

        lv.setAdapter(adapter);

        registerForContextMenu(lv);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
       ...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        ...
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {
        ...
    }
```

4.4 Handling selected items

When the user selects a menu item, the system calls `onContextItemSelected` method so you can perform the appropriate action. For example:

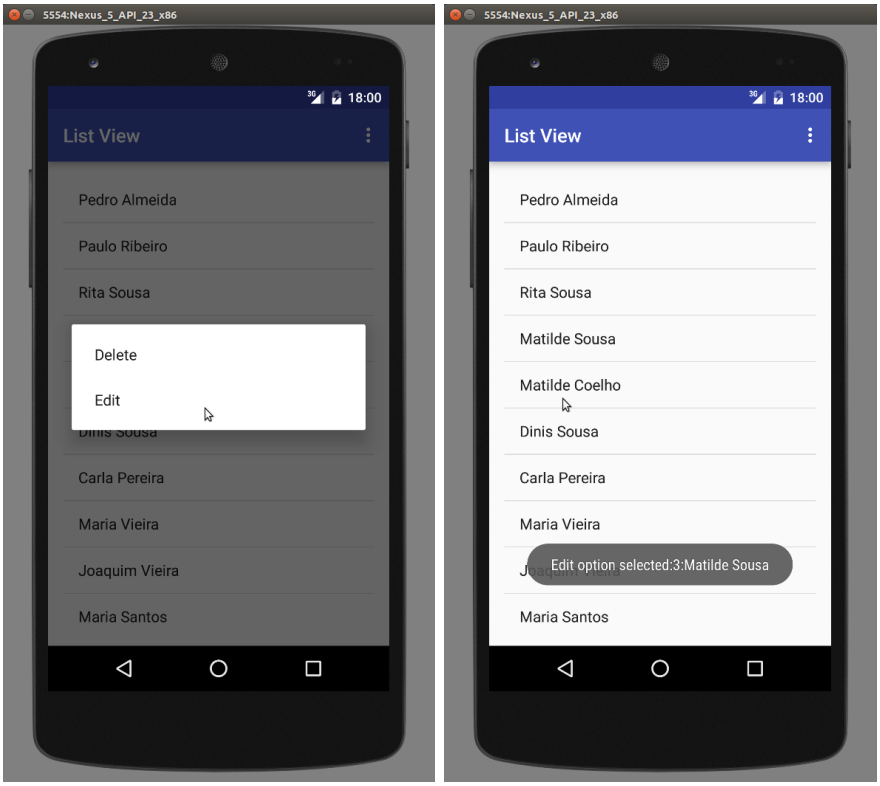
```
public class MainActivity extends AppCompatActivity {

    private ListView lv;
    private ArrayList<String> list;
    private ArrayAdapter<String> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        ...
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        ...
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {
       ...
    }

    @Override
    public boolean onContextItemSelected(MenuItem item) {
        AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
        int pos = info.position;
        long id = info.id;
        String str = (String) adapter.getItem(pos);
        switch (item.getItemId()) {
            case R.id.delete:
                Toast.makeText(getApplicationContext(),
                        "Delete option selected:" + id + ":" + str,
                        Toast.LENGTH_LONG).show();
                return true;
            case R.id.edit:
                Toast.makeText(getApplicationContext(),
                        "Edit option selected:" + id + ":" + str,
                        Toast.LENGTH_LONG).show();
                return true;
            default:
                return super.onContextItemSelected(item);
        }
    }
}
```

In this example, the selected item is an item from a `ListView`. To perform an action on the selected item, the application needs to know which is list ID for the selected item (or it's position in the `ArrayAdapter`). To get the ID, the application calls `getMenuInfo` method, which returns a `AdapterView.AdapterContextMenuInfo` object that includes the list ID for the selected item.

4.5 Running

1. To run the app from IntelliJ IDEA: Click `Run` from the toolbar.

2. Make a long press on a List View Item and select an option from the context menu.


5554:Nexus_5_API_23_x86
18:00
List View
Pedro Almeida
Paulo Ribeiro
Rita Sousa
Delete
Edit
Carla Pereira
Maria Vieira
Joaquim Vieira
Maria Santos
5554:Nexus_5_API_23_x86
18:00
List View
Pedro Almeida
Paulo Ribeiro
Rita Sousa
Matilde Sousa
Matilde Coelho
Dinis Sousa
Carla Pereira
Maria Vieira
Edit option selected:3:Matilde Sousa
Maria Santos