Lab6: Data Adapters and introduction to data storage

Objective: In this lab you will explore various ways of storing data on the phone and using online resources. You will also learn how to present data in the user interface with dynamic views such as *ListView* and use the adapter to prepare data for the presentation.

1. Introduction

Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires. Your data storage options are the following (we will focus today on bolded options):

Shared Preferences
Store private primitive data in key-value pairs.

Internal Storage
Store private data on the device memory.

External Storage
Store public data on the shared external storage.

SQLite Databases
Store structured data in a private database.

Network Connection
Data on the web with your own network server.

2. Project setup

Create new Android project called *Lab6* for the phone/tablet that is using API level 15 and has a single blank activity called *MainActivity*.

- Add new activity to your project selecting Blank Activity from the list.
  - Use the name *SettingsActivity*.
- Modify the layout of your SettingActivity.
- Add labels and TextViews for two preferences that you will store:
  - user name (String) and
  - age (int).

- Add a button to save the setting (add it to the layout and add appropriate implementation of the *OnClickListener* to the activity)
- Use menu in the MainActivity to access the SettingsActivity.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        startActivity(new Intent(this, SettingsActivity.class));
        return true;
    }

    return super.onOptionsItemSelected(item);
}
```

3. Storing preferences

The `SharedPreferences` class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types. You can use `SharedPreferences` to save any primitive data:

- booleans,
- floats,
- ints,
- longs, and
- strings.

This data will persist across user sessions (even if your application is killed). To get a `SharedPreferences` object for your application, use one of two methods:

- getSharedPreferences() - Use this if you need multiple preferences files identified by name, which you specify with the first parameter.
- getPreferences() - Use this if you need only one preferences file for your Activity. Because this will be the only preferences file for your Activity, you don't supply a name.

To write values:

1. Call `edit()` to get a `SharedPreferences.Editor`.
2. Add values with methods such as `putBoolean()` and `putString()`.
3. Commit the new values with `commit()`

To read values, use `SharedPreferences` methods such as `getBoolean()` and `getString()`.

Modify *SettingsActivity*. Your activity should:

- Access the preference object and read the activity loads
- Save the values when the Save button is pressed

> **Question:**
> How can we warn user about unsaved changes before leaving the activity?

Reading the data (`PREFS_NAME` should be defined as a string constant in the class):

```
@Override
    protected void onCreate(Bundle state){
       super.onCreate(state);
       . . .

       // Restore preferences
       SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
       String name = settings.getString("userName", "");
       setName(name);
    }
```

Saving the data:

```
@Override
    protected void onClick(View v){

      //get values from the fields

       String mName = ((TextView)findViewById(R.id.name)).getText().toString();
      // We need an Editor object to make preference changes.
      // All objects are from android.context.Context
      SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
      SharedPreferences.Editor editor = settings.edit();
      editor.putString("name", mName);

      // Commit the edits!
      editor.commit();
    }
```

Make sure that you save both name and the age using appropriate methods.

Now test your application. Run it, save the preferences and then re-run your application and see if the preferences are preserved.

4. Internal storage

You can save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed.

To create and write a private file to the internal storage:

1. Call `openFileOutput()` with the name of the file and the operating mode. This returns a`FileOutputStream`.
2. Write to the file with `write()`.
3. Close the stream with `close()`.

For example:

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```

Add new activity to your application selecting a blank activity as a starting point. Name it *TextActivity*. Make sure that the hierarchical parent of this new activity is MainActivity. Add an EditText and a Save button to this activity.

Add new option to the main activity and link it to the TextActivity.

In onClick method save the text from the EditText to the file.

In onCreate, try to retrieve data from the file. To read a file from internal storage:

1. Call openFileInput() and pass it the name of the file to read. This returns a FileInputStream.
2. Read bytes from the file with read().
3. Then close the stream with close().

5. Lists and adapters

`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. Usually, you will use the list to implement a master-detail navigation pattern. The input to the list (items in the list) can be arbitrary Java objects. The adapter extracts the correct data from the data object and assigns this data to the *views* in the row of the `ListView`. These items are typically called the *data model* of the list. An adapter can receive data as input.

The simplest ListView implementation in your application would use a string array defined in resources. Later a reference to such array can be set as a value of the property *android:entries* of the list view in the layout file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array
        name="my_array">
        <item>item1</item>
        <item>item2</item>
        <item>item3</item>
    </string-array>
</resources>
```

```
<ListView
        android:id=”@+id/myList”
        android:layout_width=”match_parent”
        android:layout_height=”wrap_content”
        android:entries="@array/my_array"
    />
```

Add new activity (*StaticListActivity*) to your application. Make sure we can start it from the MainActivity and implement a list with a static content in this activity by adding a ListView to the activity layout and connecting it (android:entries) to the string array with items of your choice.

5.1. Adapter

An *adapter* manages the data model and adapts it to the individual entries in the widget. An adapter extends the `BaseAdapter` class. It is used to “translate” more complex objects into list items.

Every line in the widget displaying the data consists of a layout which can be as complex as you want. A typical line in a list has an image on the left side (optional) and two lines of text in the middle.

ArrayAdapter

The `ArrayAdapter` class can handle a list or array of Java objects as input. Every Java object is mapped to one row. By default, it maps the `toString()` method of the object to a view in the row layout.

You can define the ID of the view in the constructor of the `ArrayAdapter` otherwise the `android.R.id.text1` ID is used as default.

The `ArrayAdapter` class allows to remove all elements in its underlying data structure with the `clear()` method call. You can then add new elements via the `add()` method or a `Collection` via the `addAll()` method.

You can also directly modify the underlying data structure and call the `notifyDataSetChanged()` method on the adapter to notify it about the changes in data.

> **Warning:**
> If you want to change the data in your adapter, the underlying data structure must support this operation. This is, for example, the case for the *ArrayList* class, but not for arrays.

ListView with ArrayAdapter

Add new activity to your application that will be called *DynamicListActivity*. The following listing shows a layout of this activity

```
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/listview"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

The following example shows how the ListView can be used with ArrayAdapter

```
public class DynamicListActivity extends Activity {

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

    final ListView listview = (ListView) findViewById(R.id.listview);
    String[] values = new String[] { "Android", "iPhone", "WindowsMobile",
        "Blackberry", "WebOS", "Ubuntu", "Windows7", "Max OS X",
        "Linux", "OS/2", "Ubuntu", "Windows7", "Max OS X", "Linux",
        "OS/2", "Ubuntu", "Windows7", "Max OS X", "Linux", "OS/2",
        "Android", "iPhone", "WindowsMobile" };

    final ArrayList<String> list = new ArrayList<String>();
    for (int i = 0; i < values.length; ++i) {
      list.add(values[i]);
    }
    final StableArrayAdapter adapter = new StableArrayAdapter(this,
        android.R.layout.simple_list_item_1, list);
    listview.setAdapter(adapter);

    listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {

      @Override
```

```
      public void onItemClick(AdapterView<?> parent, final View view,
          int position, long id) {
        final String item = (String) parent.getItemAtPosition(position);
        view.animate().setDuration(2000).alpha(0)
            .withEndAction(new Runnable() {
              @Override
              public void run() {
                list.remove(item);
                adapter.notifyDataSetChanged();
                view.setAlpha(1);
              }
            });
      }

    });
  }

  private class StableArrayAdapter extends ArrayAdapter<String> {

    HashMap<String, Integer> mIdMap = new HashMap<String, Integer>();

    public StableArrayAdapter(Context context, int textViewResourceId,
        List<String> objects) {
      super(context, textViewResourceId, objects);
      for (int i = 0; i < objects.size(); ++i) {
        mIdMap.put(objects.get(i), i);
      }
    }
```

```
@Override
    public long getItemId(int position) {
      String item = getItem(position);
      return mIdMap.get(item);
    }

    @Override
    public boolean hasStableIds() {
      return true;
    }

  }

}
```

Note:
The *ArrayAdapter* is limited as it supports only the mapping of *toString()* to one view in the row layout. To control the data assignment and to support several views, you have to create your custom adapter implementation.

Example of the custom adapter[1]

The following code shows an implementation of a custom adapter. This adapter assumes that you have two png files (no.png and ok.png) in one of your *res/drawable* folders. The coding inflates an XML layout file, finds the relevant views in the layout and sets their content based on the input data.

```
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class MySimpleArrayAdapter extends ArrayAdapter<String> {
  private final Context context;
  private final String[] values;

  public MySimpleArrayAdapter(Context context, String[] values) {
    super(context, -1, values);
    this.context = context;
    this.values = values;
  }
```

[1] The examples of the adapter is from Vogella tutorial that is available under this url: http://www.vogella.com/tutorials/AndroidListView/article.html

```
@Override
  public View getView(int position, View convertView, ViewGroup parent) {
    LayoutInflater inflater = (LayoutInflater) context
        .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    View rowView = inflater.inflate(R.layout.rowlayout, parent, false);
    TextView textView = (TextView) rowView.findViewById(R.id.label);
    ImageView imageView = (ImageView) rowView.findViewById(R.id.icon);
    textView.setText(values[position]);
    // change the icon for Windows and iPhone
    String s = values[position];
    if (s.startsWith("iPhone")) {
      imageView.setImageResource(R.drawable.no);
    } else {
      imageView.setImageResource(R.drawable.ok);
    }

    return rowView;
  }
}
```