Lecture 5

- Bundle, onSaveInstanceState
- SharedPreferences
- Retained fragment
- SQLite
- Files

Bundle

Class Bundle can store the <key, value> pairs. Key is of type String and the value can be:

- Simple variable
- String
- Serializable objects
- Parcelable objects
- Arrays

E.g:

```
Bundle bundle = new Bundle();
bundle.putInt(“nbr”,234);
bundle.putString(“model”,”V70”);
:
int aNbr = bundle.getInt(“nbr”,-1);
String str = bundle.getString(“model”);
```

onSaveInstanceState

When the configuration changes (such as rotating the phone, language change) the Activity is restarted. Some Views may lose their contents.

To prevent this loss, those values can be stored as <key, value> pairs in a Bundle that is passed as an argument to onSaveInstanceState.

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    outState.putString("tvActivity",tvActivity.getText().toString());
    outState.putInt("ivActivity", imageResource);
    outState.putString("btnActivity",btnActivity.getText().toString());
    super.onSaveInstanceState(outState);
}
```

This Bundle is received as the input parameter of onCreate. So that it can be read when creating the Activity again.

```
protected void onCreate(Bundle savedInstanceState) {
    :
    if (bundle != null) {
        tvActivity.setText(bundle.getString("tvActivity"));
        setImage(bundle.getInt("ivActivity"));
        btnActivity.setText(bundle.getString("btnActivity"));
    }
}
```

onSaveInstanceState

Though you can also restore the state by overriding onRestoreInstanceState, which is called after onStart (only if the Bundle is not null)

```
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState);

    // Restore state members from saved instance
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```

SharedPreferences

With SharedPreferences you can store <key, value> for a specific Activity. Or you can store values that the entire application can access.

In an Activity call:

```
SharedPreferences sharedPreferences = getSharedPreferences(”MainActivity",
        Activity.MODE_PRIVATE);
```

In Fragments must get first the reference to the Activity:

```
SharedPreferences sharedPreferences = getActivity().getSharedPreferences("F5Fragment",
        Activity.MODE_PRIVATE);
```

When you want to store data in SharedPreferences, request an editor first:

```
SharedPreferences.Editor editor = sharedPreferences.edit();
```

And then add pairs and apply:

```
editor.putString("tvFragment",tvFragment.getText().toString());
editor.putInt("ivFragment", imageResource);
editor.putString("btnFragment", btnFragment.getText().toString());
editor.apply();
```

SharedPreferences

Store data in SharedPreferences by requesting an Editor. Then write <key, value> pairs using put<X>() and, eventually, apply():

```
Public void onPause () {
   super.onPause();
   SharedPreferences sharedPreferences = getActivity().
            getSharedPreferences("F5Fragment", Activity.MODE_PRIVATE);
   SharedPreferences.Editor editor = sharedPreferences.edit();
   editor.putString("tvFragment",tvFragment.getText().toString());
   editor.putInt("ivFragment", imageResource);
   editor.putString("btnFragment", btnFragment.getText().toString());
   editor.apply();
}
```

Retrieve the stored data using getSharedPreferences:

```
public void onResume() {
   super.onResume();
   SharedPreferences sharedPreferences = getActivity().
            getSharedPreferences("F5Fragment", Activity.MODE_PRIVATE);
   String tvFragmentText = sharedPreferences.getString("tvFragment",null);
   if(tvFragmentText!=null) {
      tvFragment.setText(tvFragmentText);
      setImage(sharedPreferences.getInt("ivFragment", 0));
      btnFragment.setText(sharedPreferences.getString("btnFragment", ""));
   }
}
```

RetainedFragment

It is possible to use a fragment for storing data. However, one should not store anything related to UI, Activity, Context and images.

Caution: While you can store any object, you should never pass an object that is tied to the Activity, such as a Drawable, an Adapter, a View or any other object that's associated with a Context. If you do, it will leak all the views and resources of the original activity instance. (Leaking resources means that your application maintains a hold on them and they cannot be garbage-collected, so lots of memory can be lost.)

Data is stored in the fragment instance variables. The fragment has no UI. The fragment should have setters and getters.

```
public class DataFragment extends Fragment {
    private String tvActivityStr;
    private int imageResource;
    private String btnActivityStr;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }

    // setters and getters
}
```

RetainedFragment - Activity

Locate the DataFragment with the stored information during onCreate:

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

private void initDataFragment() {
    // find the retained fragment on activity restarts
    FragmentManager fm = getFragmentManager();
    dataFragment = (DataFragment) fm.findFragmentByTag("data");
    // create the fragment and add it to Activity the first time
    if (dataFragment == null) {
        // add the fragment
        dataFragment = new DataFragment();
        fm.beginTransaction().add(dataFragment, "data").commit();
    }
}
```

RetainedFragment - Activity

Update Views during onCreate or onResume. Retrieve the info from the DataFragment.

```
tvActivity.setText(dataFragment.getTvActivityStr());
ivActivity.setImageResource(dataFragment.getImageResource());
btnActivity.setText(dataFragment.getBtnActivityStr());
```

The DataFragment must be updated every time the related data in the application changes. It may be appropriate to have specific methods to do so:

```
public void setImage(int imageResource) {
    dataFragment.setImageResource(imageResource);
    ivActivity.setImageResource(imageResource);
}

public void setTvActivityText(String str) {
    dataFragment.setTvActivityStr(str);
    tvActivity.setText(str);
}

public void setBtnActivityText(String str) {
    dataFragment.setBtnActivityStr(str);
    btnActivity.setText(str);
}
```

RetainedFragment - Activity

Remove the data fragment when the Activity is being paused for good:

```
@Override
public void onPause() {
    // perform other onPause related actions
    ...
    // this means that this activity will not be recreated now, user is leaving it
    // or the activity is otherwise finishing
    if(isFinishing()) {
        FragmentManager fm = getFragmentManager();
        // we will not need this fragment anymore, this may also be a good place to signal
        // to the retained fragment object to perform its own cleanup.
        fm.beginTransaction().remove(mDataFragment).commit();
    }
}
```

SQLite

Android includes a complete database, SQLite, and a framework of classes to make handling easy.

The class SQLiteOpenHelper includes support for creating and maintaining the database, as well as supplies the database itself.

Create a subclass of SQLiteHelper for every table in the database. E.g: table Whatever must lead to a class called WhateverDBHelper. Populate the class with String constants that store the column names, the table name, as well as the SQL statement to create the database.

```
public class HighscoreDBHelper extends SQLiteOpenHelper {
    public static final String TABLE_NAME = "highscore";
    public static final String COLUMN_ID = "id";
    public static final String COLUMN_NAME = "name";
    public static final String COLUMN_POINTS = "points";

    private static final String DATABASE_NAME = "highscore.db";
    private static final int DATABASE_VERSION = 1;

    // Database creation sql statement
    private static final String DATABASE_CREATE =
            "create table " + TABLE_NAME + "(" +
                    COLUMN_ID + " text not null primary key, " +
                    COLUMN_NAME + " text not null, " +
                    COLUMN_POINTS + " integer);";
```

SQLite

The subclass to SQliteOpenHelper must also contain constructors and methods to create the table and to update the table when requested:

```
public class HighscoreDBHelper extends SQLiteOpenHelper {

    public HighscoreDBHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

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

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(HighscoreDBHelper.class.getName(),
                "Upgrading database from version " + oldVersion + " to "
                        + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

SQLite

From the subclass of SQLiteOpenHelper you can obtain the reference to write to and read from the database.

```
SQLiteDatabase db = dbHelper.getWritableDatabase(); // write
SQLiteDatabase db = dbHelper.getReadableDatabase(); // read
```

When adding or updating a row, request a writable reference to the database, then instantiate ContentValues, add the necessary data and call INSERT or UPDATE:

```
private void addScore(Score score) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(HighscoreDBHelper.COLUMN_ID,score.getId());
    values.put(HighscoreDBHelper.COLUMN_NAME,score.getName());
    values.put(HighscoreDBHelper.COLUMN_POINTS,score.getPoints());
    db.insert(HighscoreDBHelper.TABLE_NAME,"",values);
}

private void updateScore(Score score) {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(HighscoreDBHelper.COLUMN_NAME,score.getName());
    values.put(HighscoreDBHelper.COLUMN_POINTS,score.getPoints());
    db.update(HighscoreDBHelper.TABLE_NAME, values,
                HighscoreDBHelper.COLUMN_ID+"="'"+score.getId()+"'", null);
}
```

SQLite

With the delete method you can remove one or more rows

```
private void deleteAll() {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    db.delete(HighscoreDBHelper.TABLE_NAME,null,null);
}
```

It is also possible to use SQL queries as strings.

For insert, update, and delete, use the method execSQL:

```
private void add100() {
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    String sql = "UPDATE " + HighscoreDBHelper.TABLE_NAME + " " +
                 "SET " + HighscoreDBHelper.COLUMN_POINTS + "=" +
                     HighscoreDBHelper.COLUMN_POINTS + "+100 " +
                 "WHERE " + HighscoreDBHelper.COLUMN_POINTS + "<2500";
    db.execSQL(sql);
}
```

SQLite

Read data from the database calling a query method or a rawQuery method. This last one takes a SQL statement as a string input argument.

Results are stored in a Cursor object, that provides methods to retrieve the resulting data.

```
private Score[] getAllScores() {
    int idIndex, nameIndex, pointsIndex;

    SQLiteDatabase db = dbHelper.getReadableDatabase();
    Cursor cursor = db.rawQuery("SELECT * FROM " + HighscoreDBHelper.TABLE_NAME +
            " ORDER BY " + HighscoreDBHelper.COLUMN_POINTS,null);

    Score[] scores = new Score[cursor.getCount()];
    idIndex = cursor.getColumnIndex(HighscoreDBHelper.COLUMN_ID);
    nameIndex = cursor.getColumnIndex(HighscoreDBHelper.COLUMN_NAME);
    pointsIndex = cursor.getColumnIndex(HighscoreDBHelper.COLUMN_POINTS);

    for(int i=0; i<scores.length; i++) {
        cursor.moveToPosition(i);
        scores[i] = new Score(cursor.getString(idIndex),
                cursor.getString(nameIndex),
                cursor.getInt(pointsIndex));
    }
    return scores;
}
```

Files

Use files to store private information for your app. Use them as you would do it in Java.

Get a File with

public abstract FileOutputStream openFileOutput(filename, mode)

Then open a FileOutputStream to this File in PRIVATE mode:

```
FileOutputStream fos = activity.openFileOutput(filename, Context.MODE_PRIVATE);
```
or

```
FileOutputStream fos = activity.openFileOutput(filename, Context.MODE_APPEND);
```

The first command creates a file that you write each time the file is opened.

The other one opens an existing a file which you can append data to.

Finally, instantiate a proper writer object to the FileOutputStream and write.

```
ObjectOutputStream oos = new ObjectOutputStream( fos );

oos.writeObject( score );
oos.writeInt( 23 );
oos.flush();
```

Files

Use

public abstract FileInputStream openFileInput(filename)

To open a FileInputStream to a File given its filename.

```
FileInputStream fis = activity.openFileInput(filename);
```

Finally, create an appropriate InputStream to read from the FileInputStream, e.g., Object Input Stream:

```
ObjectInputStream ois = new ObjectIntputStream( fis);
```

Then you can read data from the file:

```
Score id = (Score) ois.readObject();
int str = ois.readInt( 23 );
```

The reading must be done in the same order as the file is written

Always check the IOException when dealing with Files in Java.

Don’t forget to catch it.