Mobile Application Programming: Android

Data Persistence
Activities

- Apps are composed of activities
- Activities are self-contained tasks made up of one screen-full of information
- Activities start one another and are destroyed commonly
- Apps can use activities belonging to another app
ListView

- Lists data provided by an Adapter
- Use ArrayAdapter or a custom class to provide data
- Set OnItemClickListener to react to clicks on rows
Adapter Activity

- Activity class itself can implement the Adaptor interface
- Implement each method (most simple booleans) and:
  - `getCount` - return number of objects
  - `getItem` - return item that represents row
  - `getViewTypeCount` - return 1 (or more)
  - `getView` - return a view to represent the data item
  - Reuse passed view or create a new view
  - Fill in the view with the data from `getItem`
Adapter Activity

- ListView specifically requires a **ListAdaptor** interface
- Extends **Adapter** interface and has 2 more methods:
  - **areAllItemsEnabled** - Indicates whether all the items in this adapter are enabled
  - **isEnabled**(int position) - Returns true if the item at the specified position is not a separator
- Are used to put decoration rows into the adapter
- Just return true from both methods to allow the row to be selected using a tap, raising onItemClickListener events
Key-Value Storage

Write

SharedPreferences preferences = getPreferences(MODE_PRIVATE);
SharedPreferences.Editor editor = preferences.edit();
editor.putInt("Age", age);
editor.commit();

Read

SharedPreferences preferences = getPreferences(MODE_PRIVATE);
int age = sharedPref.getInt("Age", 20 /*default age*/);

Reads / Writes: boolean, float, int, long, String
File System Access

- `getFilesDir()`
- `fileList()`
- `deleteFile(File file)`
- `getDir(String name)`

FileOutputStream os = openFileOutput("file.dat", MODE_PRIVATE);

or

File file = new File(getFilesDir(), "file.dat");
FileOutputStream os = new FileOutputStream(file, MODE_PRIVATE);

<manifest ...

<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
Flat Data Files

FileOutputStream os = openFileOutput("file.dat", MODE_PRIVATE);
FileInputStream is = openFileInput("file.dat");
Flat Data Files

```java
try {
    FileOutputStream os = openFileOutput("file.dat", MODE_PRIVATE);
    BufferedOutputStream output = new BufferedOutputStream(os);
    output.write(0xDE);
    output.write(0xAD);
    output.write(0xBE);
    output.write(0xEF);
    output.close();
}
catch (Exception e) {
    e.printStackTrace();
}
```
try {
    File file = getFileStreamPath("A Tale of Two Cities.txt");
    FileWriter writer = new FileWriter(file);
    BufferedWriter output = new BufferedWriter(writer);
    output.write("It was the best of times, ");
    output.write("it was the worst of times, ");
    //...
    output.close();
} catch (Exception e) {
    e.printStackTrace();
}
ArrayList<ArrayList<Float>> locations = null;
// Fill in collection...

try {
    FileOutputStream os = openFileOutput("file.dat", MODE_PRIVATE);
    ObjectOutputStream output = new ObjectOutputStream(os);
    output.writeObject(locations);
    output.close();
}
catch (Exception e) {
    e.printStackTrace();
}

Objects must implement Serializable interface!
import java.io.Serializable;

public class Vector implements Serializable {
    public float x;
    public float y;

    float magnitude() {
        return (float)Math.sqrt(x * x + y * y);
    }

    //...
}
JSON Using GSON
JSON Using GSON

Write

```java
Vector v = new Vector(10.0f, 20.0f);
Gson gson = new Gson();
String s = gson.toJson(v);
```

Read

```java
Vector v2 = gson.fromJson(s, Vector.class);
```

Reads / Writes: **anything!**

(String still needs to be written to file or network connection)
**Generic** type information is lost during writing

```java
class Foo<T> {
    T value;
}
Gson gson = new Gson();
Foo<Bar> foo = new Foo<Bar>();
gson.toJson(foo); // May not serialize foo.value correctly
gson.fromJson(json, foo.getClass()); // Fails read foo.value as Bar
```

Can rectify using the **TypeToken** class

```java
Type fooType = new TypeToken<Foo<Bar>>() {}.getType();
gson.toJson(foo, fooType);
Foo<Bar> foo2 = gson.fromJson(json, fooType);
```
JSON Using GSON

- **Collection** type information is also be lost during writing.

```java
Gson gson = new Gson();
Collection<Integer> ints = Lists.immutableList(1,2,3,4,5);
Type collectionType = new TypeToken<Collection<Integer>>(){}.getType();
Collection<Integer> ints2 = gson.fromJson(json, collectionType);
```

- Can rectify using the **TypeToken** class as well.