Android: Managing Persistent Data

CSC207 Winter 2015

Computer Science
UNIVERSITY OF TORONTO

Persistent Data

Android devices have two types of storage: internal and external.

We'll use internal storage for simplicity:

- Unlike external storage, by default, it is always available.
- The data saved to internal storage is only accessible by the app that saved it.
- But, when the user uninstalls the app, the data is deleted.

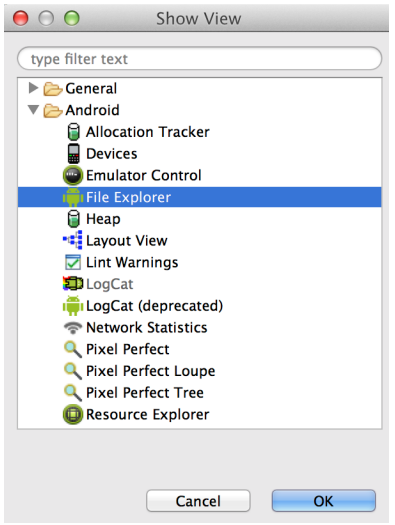
Modes

The file can be opened using one of four modes:

- `MODE_PRIVATE:` creates a new file if it doesn't exist or overwrites existing file
- `MODE_APPEND:` creates a new file if it doesn't exist or appends to an existing file
- `MODE_WORLD_READABLE:` makes the file readable by any other application
- `MODE_WORLD_WRITEABLE:` makes the file writable by any other application

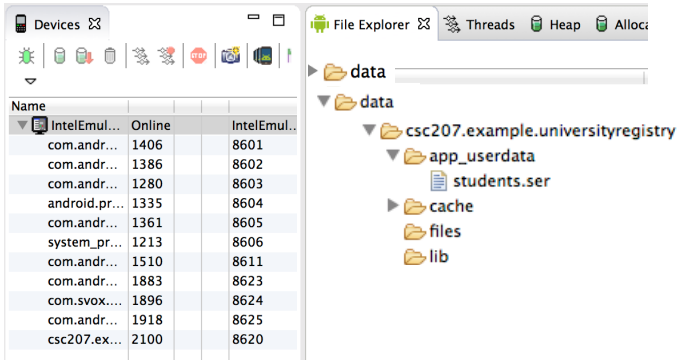
File Explorer

```
Window ->
Show View ->
other ->
Android ->
File Explorer
```



Internal Storage

```
File userdata =
  this.getApplicationContext().getDir("userdata", MODE_PRIVATE);
File studentsFile = new File(userdata, "students.ser");
```



Use `push` and `pull` to transfer files into and out of Android application internal storage.

Calling `StudentManager` Methods

MainActivity:

- `StudentManager` instance variable
- in `onCreate`, call the `StudentManager` constructor, and initialize the instance variable
- in `registerStudent`, add the new `Student` object to the `StudentManager`

Modifying the Design

Make `StudentManager` serializable, so it can be passed from one `Activity` to another.

Rather than save in `MainActivity`, add a save button to `DisplayActivity` and write to the file when that button is clicked.

Project structure

We have a large collection of classes. Some of these classes represent various types of data, others deal with the GUI, others with managing the data.

We organize the classes in a collection of packages.

In our example:

- university: contains `Student`, `Grade`, etc.
- managers: contains `StudentManager`
- GUI: contains the `Activity` classes

Benefits of this design

- provides structure
- easier to understand, easier to locate code
- separation of concerns: each package addresses a separate concern/aspect of the program
- facilitates code re-use:

  use a different GUI (keep the other packages)

  use more efficient data structures

  (keep the front-end)

  use a different mechanism for persistent storage

  (keep the data and the GUI)
- facilitates testing:

  can use regular unit tests for all the logic in the back-end

Extending the design

We have more data to manage, so we need more `Managers`.

What are the possible design options?

Create a generic `Manager` class
-parametrized by the type of data it manages

Create a `Manager` interface
-specific `Managers` implement this interface

Use inheritance
-specific `Managers` are child classes of a `Manager` class
-can even define a hierarchy of `Managers`

Which option?

Create a generic `Manager` class, parametrized by the type of data it manages.

- a good idea if all `Managers` share exactly the same behaviours
- the only difference is the type of the data they manage

```
class Manager<T> { ... }

Manager<Student> studentManager = new Manager<Student>(...);

Manager<Course> courseManager = new Manager<Course>(...);

...
```

Which option?

Use inheritance:

- a good idea if `Managers` share some behaviours
- shared behaviours belong in a parent class
- specific behaviours belong in child classes

```
class Manager { ... }

class StudentManager extends Manager { ... }

class CourseManager extends Manager { ... }

Manager<Student> studentManager = new Manager<Student>(...);

Manager<Course> courseManager = new Manager<Course>(...);

...
```

Which option?

Create a `Manager` interface:

- specific `Manager`s implement the interface
- a good idea if the `Manager`s share what they do, but not how they do it

```
interface Manager { ... }

class StudentManager implements Manager { ... }

class CourseManager implements Manager { ... }

Manager studentManager = new StudentManager(...);

Manager courseManager = new CourseManager(...);
...
```