Android “Master” Class 2013

Lecture 3, Octover 31. 2013

Agenda

- Finish up Fragments
- Dialogs
- Multithreading on Android
- HTTP/REST Introduction
- SQLite Introduction

Agenda

- **Finish up Fragments**
- Dialogs
- Multithreading on Android
- HTTP/REST Introduction
- SQLite Introduction

Fragments

- Assume we have this Fragment class:

```
public class MyFragment extends Fragment {
        // Each instance should have a unique id that I specify
        private int mId;

        public MyFragment(int id) {
                mId = id;
        }
}
```

Fragments

- Assume we have this Fragment class:

```
public class MyFragment extends Fragment {
        // Each instance should have a unique id that I specify
        private int mId;

        // Will this work?
        public MyFragment(int id) {
                mId = id;
        }
}
```

Fragments

- Fragments are recreated on configuration change

Fragments

- Fragments are recreated on configuration change
- Their internal state is saved (not member variables), then a new instance is created using that saved state

Fragments

- Fragments are recreated on configuration change
- Their internal state is saved (not member variables), then a new instance is created using that saved state
- Use onSaveInstanceState(Bundle outState) to save

Fragments

- Fragments are recreated on configuration change
- Their internal state is saved (not member variables), then a new instance is created using that saved state
- Use onSaveInstanceState(Bundle outState) to save
- Use onActivityCreated(Bundle savedInstanceState) to restore

Fragments

- Quick note on savedInstanceState Bundle

Fragments

- Quick note on savedInstanceState Bundle
  - Bundle can only store data that can be marshalled across processses (i.e. primative types: int, char, boolean, etc.)

Fragments

- Quick note on savedInstanceState Bundle
  - Bundle can only store data that can be marshalled across processses (i.e. primative types: int, char, boolean, etc.)
  - Can also store Parcelable objects

Fragments

- Quick note on savedInstanceState Bundle
  - Bundle can only store data that can be marshalled across processses (i.e. primative types: int, char, boolean, etc.)
  - Can also store Parcelable objects
  - A parcelable object is an object whose member variables are all also parcelable (primitive types are, by definition, parcelable)

Fragments

- Quick note on savedInstanceState Bundle
  - Bundle can only store data that can be marshalled across processses (i.e. primative types: int, char, boolean, etc.)
  - Can also store Parcelable objects
  - A parcelable object is an object whose member variables are all also parcelable (primitive types are, by definition, parcelable)

Fragments

- Quick note on savedInstanceState Bundle
  - Bundle can only store data that can be marshalled across processses (i.e. primative types: int, char, boolean, etc.)
  - Can also store Parcelable objects
  - A parcelable object is an object whose member variables are all also parcelable (primitive types are, by definition, parcelable)
  - Member variables are destroyed along with an instance of their enclosing class

Fragments

- Quick note on savedInstanceState Bundle
  - Bundle can only store data that can be marshalled across processses (i.e. primative types: int, char, boolean, etc.)
  - Can also store Parcelable objects
  - A parcelable object is an object whose member variables are all also parcelable (primitive types are, by definition, parcelable)
  - Member variables are destroyed along with an instance of their enclosing class
  - However, a copy of each one can be saved into a Bundle and be used to restore the state into a new instance

Fragments

```
// … MyFragment …
private static final String KEY_ID = “id”;
@override
public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putInt(KEY_ID, mId);
}
```

Fragments

- As a result of this, FragmentManager must save a Fragment's state, create a new one, and pass the saved state to the new instance

Fragments

- As a result of this, FragmentManager must save a Fragment's state, create a new one, and pass the saved state to the new instance
- This introduces a restriction:

Fragments

- As a result of this, FragmentManager must save a Fragment's state, create a new one, and pass the saved state to the new instance
- This introduces a restriction:
  - FragmentManager only creates new instances with default constrcutor, i.e. MyFragment()

Fragments

- How do we pass arguments to a Fragment we create?

Fragments

- How do we pass arguments to a Fragment we create?
- Luckily, setArguments(Bundle args) is supplied

Fragments

- How do we pass arguments to a Fragment we create?
- Luckily, setArguments(Bundle args) is supplied
- Static “create” method that takes args, bundles them up, places them in a new Fragment instance, and returns the instance

Fragments

```
// … MyFragment …
public static MyFragment create(int id) {
        MyFragment f = new MyFragment();
        Bundle args = new Bundle();
        args.putInt(KEY_ID, id);
        f.setArguments(args);
        return f;
}
```

Fragments

```
// … MyFragment …
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mId = getArguments().getInt(KEY_ID);
}
```

Fragment

- Arguments bundle is saved across configuration change, so in this toy example, the override to onSaveInstanceState is unnecessary

Fragment

- Arguments bundle is saved across configuration change, so in this toy example, the override to onSaveInstanceState is unnecessary
- Use when you want to save something that could change after initialization, like a count or displayed message

- This goes for Activities as well

Agenda

- **Finish up Fragments**
- **Dialogs**
- Multithreading on Android
- HTTP/REST Introduction
- SQLite Introduction

Dialogs

- Nice way of getting quick user input

Dialogs

- Nice way of getting quick user input
- Or simply just notify the user

Dialogs

- Pre-Fragments method: Use AlertDialog.Builder

Dialogs

```
AlertDialog.Builder builder =
        new AlertDialog.Builder(getActivty());
builder.setMessage("Are you a human?")
        .setPositiveButton("Yes", new OnClickListener() {
                public void onClick(DialogInterface d, int id) {
                        d.dismiss();
                }
        })
        .setNegativeButton("No", new OnClickListener() {
                public void onClick(DialogInterface d, int id) {
                        d.cancel();
                }
        }
        .create()
        .show();
```

Dialogs

- Could be called anywhere from within the Activity

Dialogs

- Could be called anywhere from within the Activity
- Downside: Not maintained across configuration changes

Dialogs

- Could be called anywhere from within the Activity
- Downside: Not maintained across configuration changes
  - Required saving and restoring a flag in the savedInstanceState and showing a new dialog if the flag were set upon restore

Dialogs

- Use DialogFragment instead

Dialogs

- Use DialogFragment instead
- Wrapper around the DialogInterface that has the same lifecycle as a normal fragment

Dialogs

- Use DialogFragment instead
- Wrapper around the DialogInterface that has the same lifecycle as a normal fragment
- New callback: onCreateDialog()

Dialogs

- Use DialogFragment instead
- Wrapper around the DialogInterface that has the same lifecycle as a normal fragment
- New callback: onCreateDialog()
- Called after onCreate, but before onCreateView

Dialogs

- Use DialogFragment instead
- Wrapper around the DialogInterface that has the same lifecycle as a normal fragment
- New callback: onCreateDialog()
- Called after onCreate, but before onCreateView
- Use onCreateView to inflate a custom view from xml to be used as dialog's view

Dialogs

- Code example

Agenda

- **Finish up Fragments**
- Dialogs
- **Multithreading on Android**
- HTTP/REST Introduction
- SQLite Introduction

Multithreading on Android

- A thread is a unit of execution in a process (application)

Multithreading on Android

- A thread is a unit of execution in a process (application)
- Think of a process as a play and each actor as an actor in that play fighting for stage time (resources)*

* Analogy credit Professor Peter Chen

Multithreading on Android

- A thread is a unit of execution in a process (application)
- Think of a process as a play and each actor as an actor in that play fighting for stage time (resources)*
- Each thread can be doing its own thing while the others work

* Analogy credit Professor Peter Chen

Multithreading on Android

- A thread is a unit of execution in a process (application)
- Think of a process as a play and each actor as an actor in that play fighting for stage time (resources)*
- Each thread can be doing its own thing while the others work
- Each thread can run on a different CPU to get this effect

* Analogy credit Professor Peter Chen

Multithreading on Android

- A thread is a unit of execution in a process (application)
- Think of a process as a play and each actor as an actor in that play fighting for stage time (resources)*
- Each thread can be doing its own thing while the others work
- Each thread can run on a different CPU to get this effect
  - Can be simulated on a single CPU by switching **contexts**

* Analogy credit Professor Peter Chen

Multithreading on Android

- Why is this important?

Multithreading on Android

- Why is this important?
  - Each app uses a single thread by default: the main (UI) thread

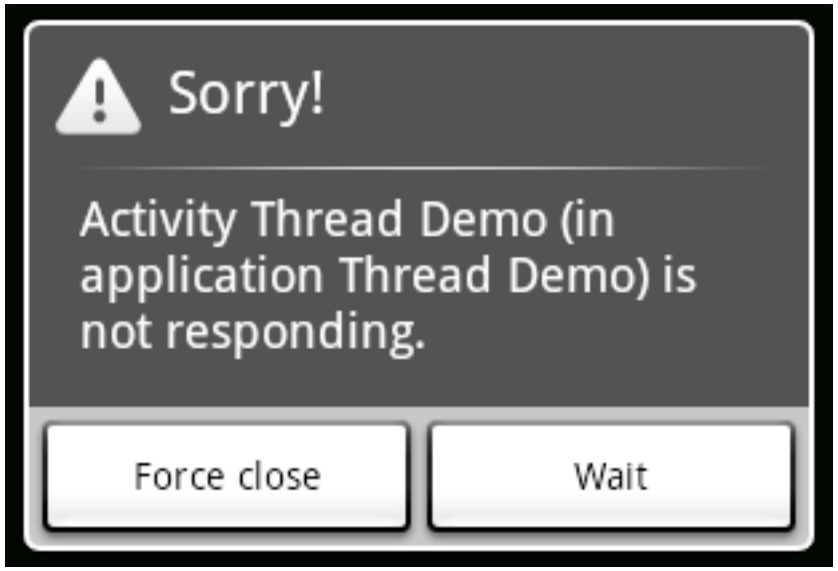
Multithreading on Android

- Some operations can block, waiting for input from some other module, or even another device
  - Ex. HTTP, disk reads/writes, database transactions, user input

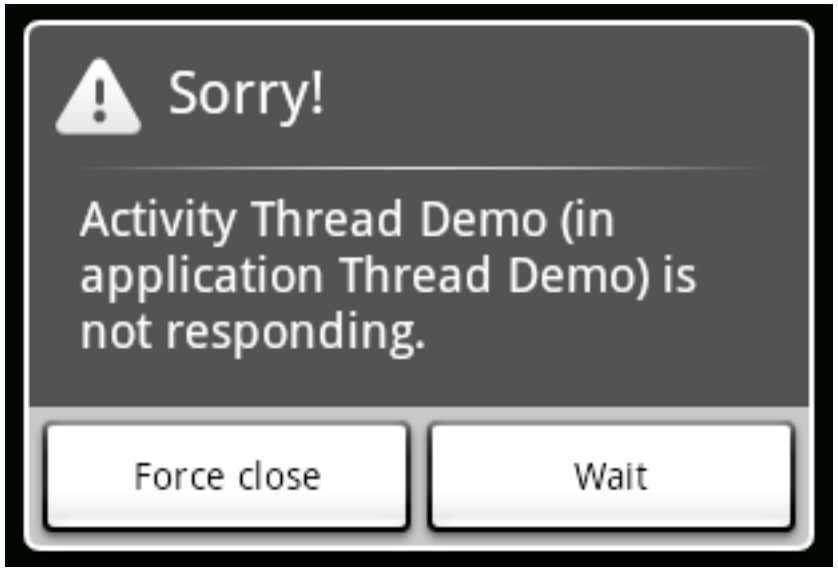
Multithreading on Android

- Some operations can block, waiting for input from some other module, or even another device
  - Ex. HTTP, disk reads/writes, database transactions, user input
- If any of these operations are carried out in the context of the main thread, the user's experience could be interrupted

Multithreading on Android

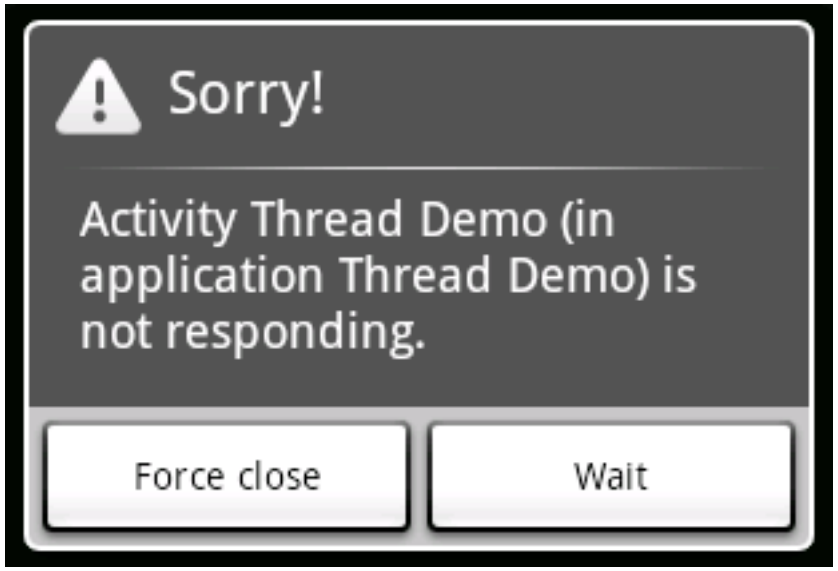


Multithreading on Android



- If the main thread of you application is blocked for 5 seconds or more, this message is shown

Multithreading on Android



- If the main thread of you application is blocked for 5 seconds or more, this message is shown
  - The dreaded ANR (app not responding)!
  - AVOID THIS AT ALL COSTS

Multithreading on Android

- When might this happen?

Multithreading on Android

- When might this happen?
  - HTTP request that takes a while for the server to respond to
  - Trying to write to a file that is locked because a different application is currently writing to it
  - A database query gets hung up for some reason (rare)
  - Looping on user input from the main thread

Multithreading on Android

- How can we avoid this?

Multithreading on Android

- How can we avoid this?
- Spawn our own background threads for blocking and expensive operations

Multithreading on Android

- Android (and Java by extension) offers many facilities this:

Multithreading on Android

- Android (and Java by extension) offers many facilities this:
  - Thread

Multithreading on Android

- Android (and Java by extension) offers many facilities this:
  - Thread
  - ThreadPoolExecutor

Multithreading on Android

- Android (and Java by extension) offers many facilities this:
  - Thread
  - ThreadPoolExecutor
  - AsyncTask

Multithreading on Android

- Android (and Java by extension) offers many facilities this:
  - Thread
  - ThreadPoolExecutor
  - AsyncTask
  - Loader

Multithreading on Android

- Android (and Java by extension) offers many facilities this:
  - Thread
  - ThreadPoolExecutor
  - AsyncTask
  - Loader
  - Handler + HandlerThread

Multithreading on Android

- Android (and Java by extension) offers many facilities this:
  - Thread
  - ThreadPoolExecutor
  - AsyncTask
  - Loader
  - Handler + HandlerThread
  - Service

Multithreading on Android

- Android (and Java by extension) offers many facilities this:
  - Thread
  - ThreadPoolExecutor
  - AsyncTask
  - Loader
  - Handler + HandlerThread
  - Service…?

Multithreading on Android

- Android (and Java by extension) offers many facilities this:
  - Thread
  - ThreadPoolExecutor
  - AsyncTask
  - Loader
  - Handler + HandlerThread
  - ~~Service –~~ Runs in the context of the main thread!

Multithreading on Android

- Android (and Java by extension) offers many facilities this:
  - Thread
  - ThreadPoolExecutor
  - AsyncTask
  - Loader
  - Handler + HandlerThread
  - ~~Service –~~ Runs in the context of the main thread!
    - IntentService is a Service that maintains a background automatically

Multithreading on Android

- We'll go into details on each one later, as they each have their own advantages/disadvantages

Agenda

- Finish up Fragments
- Dialogs
- Multithreading on Android
- **HTTP/REST Introduction**
- SQLite Introduction

HTTP/REST Introduction

- HTTP: HyperText Transfer Protocol

HTTP/REST Introduction

- HTTP: HyperText Transfer Protocol
- One of the most popular way to communicate with a server

HTTP/REST Introduction

- REST: REpresentational State Transfer

HTTP/REST Introduction

- REST: REpresentational State Transfer
- Offers common methods for taking actions on a web server

HTTP/REST Introduction

- REST: REpresentational State Transfer
- Offers common methods for taking actions on a web server:
  - GET – Retrieve existing data

HTTP/REST Introduction

- REST: REpresentational State Transfer
- Offers common methods for taking actions on a web server:
  - GET – Retrieve existing data
  - POST – Add new data

HTTP/REST Introduction

- REST: REpresentational State Transfer
- Offers common methods for taking actions on a web server:
  - GET – Retrieve existing data
  - POST – Add new data
  - PUT – Update existing data

HTTP/REST Introduction

- REST: REpresentational State Transfer
- Offers common methods for taking actions on a web server:
  - GET – Retrieve existing data
  - POST – Add new data
  - PUT – Update existing data
  - DELETE – Remove existing data

HTTP/REST Introduction

- REST: REpresentational State Transfer
- Offers common methods for taking actions on a web server:
  - GET – Retrieve existing data
  - POST – Add new data
  - PUT – Update existing data
  - DELETE – Remove existing data
- There are others, but these are the main ones we will focus on

HTTP/REST Introduction

- GET myserver.com/home.html HTTP/1.1
  - Requests the html for the home page of my website

HTTP/REST Introduction

- GET myserver.com/home.html HTTP/1.1
  - Requests the html for the home page of my website
- POST myserver.com/users HTTP/1.1
  user=ajkause&password=12345678
  - Add a new user to my server

HTTP/REST Introduction

- GET myserver.com/home.html HTTP/1.1
  - Requests the html for the home page of my website
- POST myserver.com/users HTTP/1.1
  user=ajkause&password=12345678
  - Add a new user to my server
- PUT myserver.com/users/name HTTP/1.1
  first=AJ&last=Kause
  - Update a users's name on the server

HTTP/REST Introduction

- GET myserver.com/home.html HTTP/1.1
  - Requests the html for the home page of my website
- POST myserver.com/users HTTP/1.1
  user=ajkause&password=12345678
  - Add a new user to my server
- PUT myserver.com/users/name HTTP/1.1
  first=AJ&last=Kause
  - Update a users's name on the server
- DELETE myserver.com/users HTTP/1.1
  user=ajkause
  - Removes a user from my server

Agenda

- Finish up Fragments
- Dialogs
- Multithreading on Android
- HTTP/REST Introduction
- **SQLite Introduction**

SQLite Introduction

- SQL: Structured Query Language
- Used for database operations

SQLite Introduction

- Offers methods for easy database manipulation:

SQLite Introduction

- Offers methods for easy database manipulation:
- SELECT – query the db

SQLite Introduction

- Offers methods for easy database manipulation:
- SELECT – query the db
- INSERT – Insert into the db

SQLite Introduction

- Offers methods for easy database manipulation:
- SELECT – Query existing records in thd db
- INSERT – Insert a new record into the db
- UPDATE – Update an existing record into the db

SQLite Introduction

- Offers methods for easy database manipulation:
- SELECT – Query existing records in thd db
- INSERT – Insert new records into the db
- UPDATE – Update existing records into the db
- DELETE – Remove records from the db

SQLite Introduction

- Offers methods for easy database manipulation:
- SELECT – Query existing records in thd db
- INSERT – Insert new records into the db
- UPDATE – Update existing records into the db
- DELETE – Remove records from the db
- CREATE – Creates a new table in the db

SQLite Introduction

```
CREATE TABLE users
        (_id INTEGER PRIMARY KEY AUTOINCREMENT,
         username TEXT NOT NULL,
         birthday TEXT);
```

SQLite Introduction

SELECT * FROM users WHERE username=‘aj’;

INSERT INTO users VALUES(0,‘aj’,’12345’);

UPDATE users SET password=‘899100’ WHERE username=‘aj’;

DELETE users WHERE username=‘aj’;

Thank You

Q&A until 9pm