(1) SwingWorker
(2) Intro to Android

17 March 2011
CMPT166
Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Outline for today

- SwingWorker class for threads in Swing
  - Sending and receiving results:
    - doInBackground() and done()
  - Publishing progress updates / interim results:
    - publish() and process()
  - Cancelling a background task
    - cancel() and isCancelled()
- Intro to Android
  - Android SDK + ADT plug-in for Eclipse

Threads in Swing

- Swing programs have multiple threads:
  - Init thread (main() setup before GUI)
  - Event dispatch thread (interacts w/GUI)
  - Any worker threads you create
- Only the event dispatch thread should access the GUI (change widget text, etc.)
  - Worker threads have to ask the event dispatch thread to update the GUI
- How do worker threads communicate to the event dispatch thread?

SwingWorker abstract class

- Subclass of Thread that allows you to:
  - Define the task to be done in background
  - Run code on the event dispatch thread when the worker thread is done
  - Return an object from the worker thread to the event dispatch thread
  - Send progress updates from the worker thread to the event dispatch thread
  - Define bound properties: when modified, event dispatch thread receives an event

Using SwingWorker

- SwingWorker is abstract: so subclass it

```
class Fetcher extends SwingWorker {
```

- SwingWorker is generic: specify the class of the result that the bg task will return:

```
class Fetcher extends SwingWorker<Image, Void>
```

- Define the task in doInBackground():

```
public Image doInBackground() { ... }
```

  - Return type must be same as in template
  - Should only modify local variables
  - Return result of the long-running task

Getting the result: done()

- Override the done() method to define how the event dispatch thread gets the results:

```
public void done() {
try {
    myButton.setIcon( get() );
} except (InterruptedException e) {
} except (ExecutionException e) {
}
```

- Will be run on the event dispatch thread
  - When worker thread has finished
- get() returns result of doInBackground()

Starting the worker thread

- Create an instance of your subclass of SwingWorker and call its .execute() method

```
Fetcher fetcher = new Fetcher();
fetcher.execute();
```

  - Equivalent to the usual Thread.start()
- E.g., in a button's action listener:

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Fetcher fetcher = new Fetcher();
        fetcher.execute();
    }
});
```

Putting it together

```
button.addActionListener( new ActionListener() {
    void actionPerformed(ActionEvent evt) {
        (new SwingWorker<ImageIcon, Void>() {
            public ImageIcon doInBackground() {
                ImageIcon img =
                        (ImageIcon) serverIn.getObject();
                return img;
            }
            public void done() {
                try {
                    myButton.setIcon( get() );
                } except (InterruptedException e) {
                } except (ExecutionException e) {
                }
            }
        } ).execute();
    } } );
```

event listener for button

anonymous class

slow task

get obj returned by doInBackground()

run by event disp. thread

start the thread

Publishing progress updates

- The worker thread may send objects to the event dispatch thread as interim results:
- Declare type of interim result in template:

```
... extends SwingWorker<Image, Float> {
```

- From doInBackground(), call publish():

```
publish( bytesFetched / totBytes );
```

- Override process() to specify how event dispatch thread handles an update:

```
public void process( List<Float> updates ) {
```

  - Parameter is a List of accumulated updates
- publish() may be called very very frequently!

Summary of SwingWorker

```
(new SwingWorker<ResultType, UpdateType>() {
    public ResultType doInBackground() {
        // long task
        // periodically call publish() with an update
        // return result
    }
    public void process( List<UpdateType> updates ) {
        // update progress bar UI, etc.
    }
    public void done() {
        try {
            // get() result and update UI
        } except (InterruptException e) { ... }
    }
}).execute();
```

Cancelling a background task

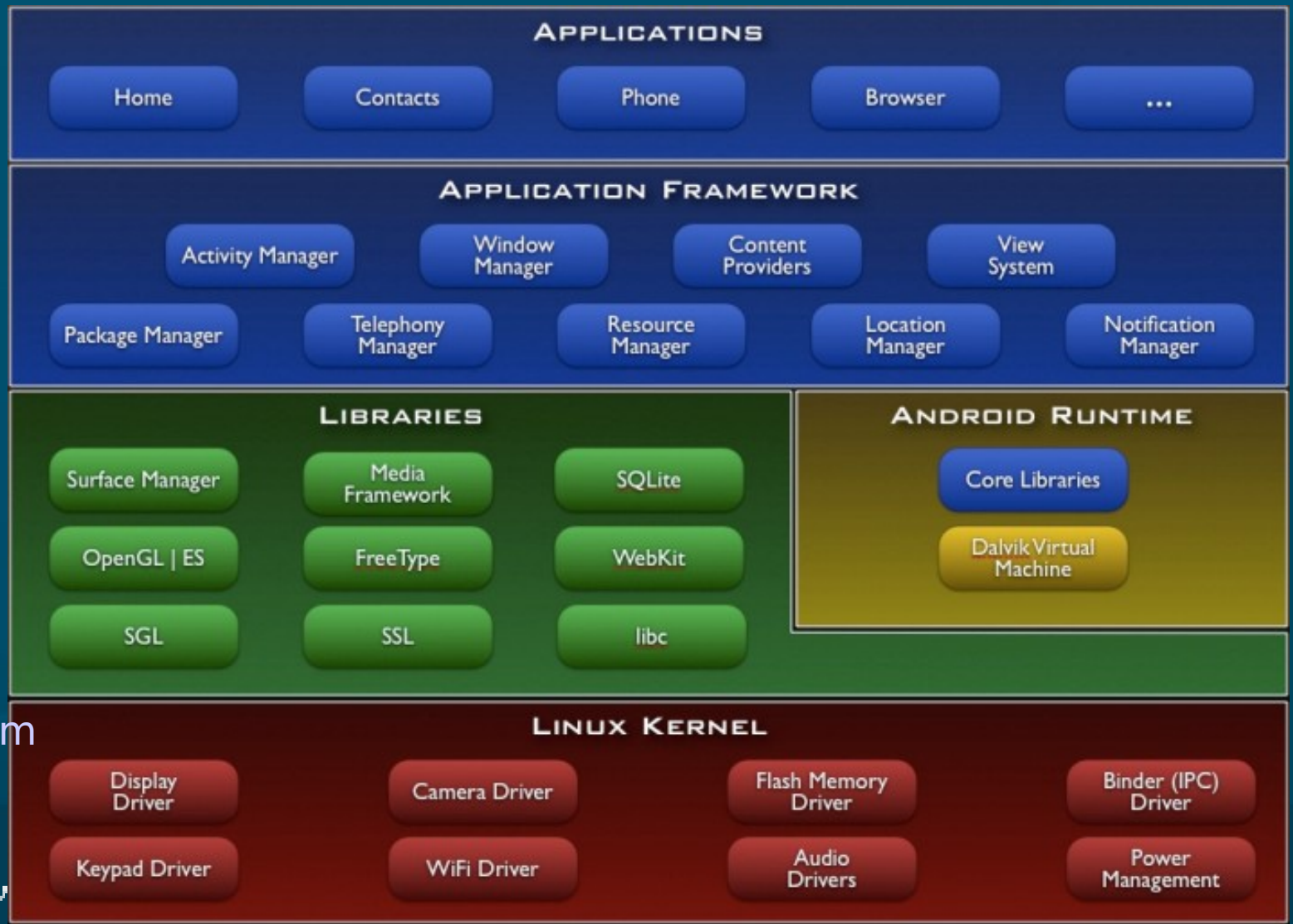
- UI thread calls worker's .cancel() method
  - Means thread can't be an anon. object
- In the worker (doInBackground()), check if we've been cancelled: if (isCancelled())
- Or cancel using interrupts:
  - Call cancel(true) instead of just cancel()
  - Worker thread receives InterruptException
  - Only if worker thread is doing something that can raise InterruptException: Thread.sleep(), network send/receive, etc.

Android OS

- Open-source mobile OS (mostly Apache licence)
- Developed by Google + Open Handset Alliance
- Linux kernel
- Most apps written in Java, using Android SDK
- Apps run on Dalvik: custom Java VM
- Android Open Source Project: fully open-source
- "Google Experience":
  adds closed-source apps (Maps, Gmail, etc.)
- Hardware drivers are also often closed-source

Android architecture

- Android is: OS, core libraries, “middleware”, plus basic applications



Source: d.android.com

Android features

- Component architecture: reuse parts of apps
- Integrate web browser into your app (WebKit)
- Audio, video, images (MPEG4, MP3, PNG, etc.)
- 2D and 3D graphics (OpenGL-ES 1.0/1.1)
- SQLite on-board database
- Telephony (calls, SMS, etc.)
- Networking: EDGE/3G, WiFi, Bluetooth
- Sensors: camera, GPS, compass, accelerometer, ...
- Develop in Eclipse, debug on phone

Getting started with Android

- Eclipse IDE for Java
- Android SDK starter package
- ADT plugin for Eclipse
- From plugin, add Android 1.6 platform
  - Could also develop for 1.5, 2.0, 2.1, etc.
  - Setup an emulator instance(virtual phone)
- Try the “Hello World!” tutorial
  - Run/debug on the emulator
  - Run/debug on actual phone via USB