CS371m - Mobile Computing

Responsiveness
An App Idea

• From Nifty Assignments
• Draw a picture use randomness
• Pick an equation at random
• Operators in the equation have the following property:
  Given an input between -1 and 1 the output is also between -1 and 1
• sin and cos scaled to pi / 2, multiply, average, remainder (except for 0)
Random Art

• The color at any given point is based on the x and y coordinates of that point scaled to between -1 and 1
• Feed the x and y coordinates into the equation
• Pick equations at random, keep the good pictures, throws away the boring ones
• Given the equation we can reproduce the image
• Color of this pixel?
• Assume large frame is 400 by 300.
• Assume this pixel is at 100, 30
• x = 100 / 400 = 0.25 -> scaled to -1 to 1 = -0.5
• y = 30 / 300 = 0.1 -> scaled to -1 to 1 = -0.8
• Plug these values into random equation:
• Assume equation is $y \times A S C S y S S x C y C A C M M S C S S C C$
  postfix, A = Average, S = Sin, C = Cos, M = Multiply
• Assume answer is 0.75. Scale to number of colors.
  Assume 256 shades of gray.
• Color at that pixel is 224$^{th}$ shade of gray (224, 224, 224)
Result  yxASCySSxCyCACyMMSyCSSCC
Result

$\text{xxACSSxCAYCyxASASCAyCCAyyyAAxMS}$

$\text{xCxCAxSySMMMCMMCSC}$
Result yCCSxxMSSAS
Results
Results
Results
RandomArt Application

• Create a subclass of View that does the computation and draws the graphics
• More on 2d graphics later in term
  – but we simply override the onDraw(Canvas) method and draw want we want
  – colors via Paint objects
  – Canvas.drawPoint(x, y, Paint) method
• add click listener to the View so click results in new expression and a redraw
  – invalidate() -> leads to onDraw(Canvas)
What happens when we run the Random Art App and create a new image?
A. Works great
B. Runtime Error
C. Freezes Device Permanently
D. Freezes Device with Error Dialog
The Problem

• Neat idea but computationally expensive
• 1080 by 1920 screen on Google Nexus 5X
• 2 million plus pixels
• depending on the expressions, tens of millions of computations, plus the rendering
Responsiveness

• user's threshold of pain? 1 second? 2?
  – Android dev documents claim 100 to 200 milliseconds (0.1 to 0.2 seconds)

• The Android Systems has its own threshold of pain
  – if the systems determines an application has become unresponsive it displays the Application Not Responding (ANR) dialog

• ANR occurs if app not responsive to user input
Android System

• The Activity Manager and Window Manager system services monitor applications for responsiveness

• ANR dialog displayed if:
  – No response to an input event such as a key press or screen touch in 5 seconds
  – A BroadcastReceiver doesn't finish executing in 10 seconds
Typical Blocking Operations

- complex calculations or rendering
  - AI picking next move in game
- looking at data set of unknown size
- parsing a data set
- processing multimedia files
- accessing network resources
- accessing location based services
- access a content provider
- accessing a local database
- accessing a file
The Main Thread

• For applications that consist of an Activity (or Activities) it is vital to **NOT** block the Main thread

• AND on API level 11 and later certain operations **must** be moved off the main UI thread
  – code that accesses resources over a network
  – for example, HTTP requests on the main UI thread result in a NetworkOnMainThreadException
  – discover StrictMode
    
The Main Thread

• When application launched system creates a thread called main aka the UI thread
• One thread for all UI components
• In charge of dispatching events to UI widgets
  – Including drawing them
  – When user touches a button on your screen, your app’s UI / Main thread dispatches the touch event to the widget
Enabling Responsiveness

- move time consuming operations to child threads
  - Android AsyncTask
  - View.post(Runnable)
  - View.postDelayed(Runnable, long)
  - Service?
- provide progress bar for worker threads
- big setups -> use a splash screen or render main view as quickly as possible and filling in information asynchronously
- assume the network is SLOW
- don't access the Android UI toolkit from outside the UI thread
  - can result in undefined and unexpected behavior
Asking for Trouble

• Loading image from network may be slow and can block the main (UI) thread of the application - (Change to TBBT app)

```java
private void setImage() {
    Bitmap b = loadImageFromNetwork("http://www.utexas.edu/sites/default/mImageView setImageBitmap(b);
}

private Bitmap loadImageFromNetwork(String imageUrl){
    Bitmap bitmap = null;
    try {
        URL url = new URL(imageUrl);
        bitmap = BitmapFactory.decodeStream((InputStream) url.getContent());
    } catch (IOException e) {
        Log.d(TAG, "problem: ");
    }
    return bitmap;

```
AsyncTask

• Android class to handle simple threading for operations that take a few seconds
• Removes some of the complexities of Java Thread, Runnables, and Android Handler classes
• UI creates an AsyncTask object and calls the `execute` method
• Result *published* to the UI thread
AsyncTask

• Three Generic Parameters
  – data type of Parameter(s) for task
  – data type of Progress
  – data type of Result

• four steps in carrying out task
  – onPreExecute()
  – doInBackground(Param... params)
  – onProgressUpdate(Progress values)
  – onPostExecute(Result result)
Methods

• `onPreExecute()` runs on UI thread before background processing begins
• `doInBackground(Param... params)` runs on a background thread and won't block UI thread
• `publishProgress(Progress... values)` method invoked by `doInBackground` and results in call to `onProgressUpdate()` method on UI thread
• `onPostExecute(Result result)` runs on UI thread once `doInBackground` is done
public void onClick(View v) {
    new DownloadImageTask().execute("http://www.utexas
}

private class DownloadImageTask
        extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to
     * perform work in a worker thread and
     * delivers it the parameters given
     * to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform
     * work in the UI thread and delivers
     * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
Random Art with AsyncTask

- Add progress bar and button for new art
- create a Bitmap and draw to that
- `<Integer, Integer, Bitmap>`
Just One More
Loaders

• Loader classes introduced in API 11
• Help asynchronously load data from content provider or network for Activity or Fragment
• monitor data source and deliver new results when content changes
• multiple classes to work with