Chapter 8.

Handling of Concurrency

2023-2024

COMP7506 Smart Phone Apps Development

Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)

Department of Computer Science, The University of Hong Kong

Agenda

- Concept of Thread
- Non-Thread Example (Not Working)
- Thread Example 1 (Not Working)
- Thread Example 2 (Working)
- Thread Example 3 (Working)
- Stopping a Thread
- Handler
- Concurrency Issues in iOS

Concept of “Thread”

Process vs. Thread

- Process
  - Basic unit of application execution
  - Don’t share stuff (e.g., state or memory) with each other
  - Context switch expensive
- Thread
  - Natural for one app to contain multiple threads
  - Share stuff (e.g., state and memory)
- Parallel computing

Main Thread

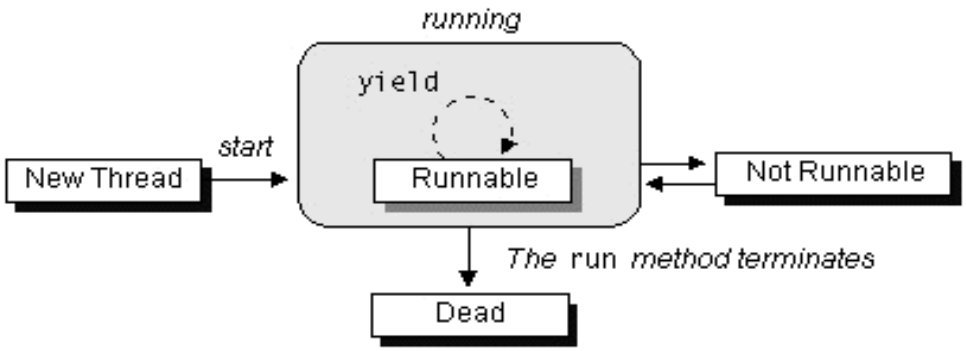
- UI thread
  - views (layouts) and viewgroups (widgets)
  - A single queue to handle all events
    - e.g. button pushing, redrawing, …
- Difficult to get things right if multi-threading UI
  - Android UI is not thread safe (what if more than one thread access the same UI?)
- Do not block UI thread
  - Long running operations can hamper UI experience
    - e.g., long downloading
    - Android will kill app with hanging UI thread
  - Use different threads

Rules for Android UI

- Do not block the UI thread
- Do not access the UI toolkit from outside the UI thread

Common Threading Class

- Thread
  - Starts a Runnable object
- Runnable
  - A class that can be instantiated by a Thread or a Handler
  - Must override 1 method: run()
  - Analogous to View.OnClickListener() in which you must override onClick()



Non-Thread Example (Not Working)

Non-Thread Example (activity_main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="hkucs.thread_test_1.MainActivity">
<Button
                android:id="@+id/Button1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Load Image"
                tools:layout_editor_absoluteY="2dp"
                tools:layout_editor_absoluteX="5dp" />
<ImageView
                android:id="@+id/ImageView1"
                android:layout_width="305dp"
                android:layout_height="407dp"
                tools:layout_editor_absoluteY="52dp"
                tools:layout_editor_absoluteX="5dp" />
</android.support.constraint.ConstraintLayout>
```

Non-Thread Example (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="hkucs.thread_test_1">

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

<application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
                <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
        </activity>
</application>
</manifest>
```

Related Knowledge – uses-permission

- Requests a permission that the application must be granted in order for it to operate correctly. Permissions are granted by the user when the application is installed (on devices running Android 5.1 and lower) or while the app is running (on devices running Android 6.0 and higher).
- Syntax: <uses-permission android:name="*string*" android:maxSdkVersion="*integer*" />
- Attributes:
  - android:name: Name of the permission
  - android:maxSdkVersion: The highest API level at which this permission should be granted to your app. Setting this attribute is useful if the permission your app requires is no longer needed beginning at a certain API level.
- You can take a look at the site below for a full list of uses-permission in Android: https://gist.github.com/Arinerron/1bcaadc7b1cbeae77de0263f4e15156f

Non-Thread Example (MainActivity.java)

```
public class MainActivity extends AppCompatActivity {
        Button mButton;
        ImageView mImageView;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                mImageView = (ImageView) findViewById(R.id.ImageView1);
                mButton = (Button) findViewById(R.id.Button1);
                mButton.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                Bitmap b = loadImageFromNetwork
                                ("https://i.cs.hku.hk/~twchim/c7506/android.png");
                                mImageView.setImageBitmap(b);
                        };
                });
        }
```

Our department's server does not support insecure connection from third-party clients (including mobile apps). Thus the following URL does not work: http://www.cs.hku.hk/~twchim/c7506/android.png .

Non-Thread Example (MainActivity.java)

```
        private Bitmap loadImageFromNetwork(String url){
                try {
                        Bitmap bitmap = BitmapFactory.decodeStream
                        ((InputStream)new URL(url).getContent());
                        return bitmap;
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return null;
        }
}
```

This program does not work most of the time. Why???

If the downloading cannot be completed within a pre-defined interval, the process will be killed by the system.

Thread Example 1 (Not Working)

With the same activity_main.xml and AndroidManifest.xml

Thread Example (MainActivity.java)

```
public class MainActivity extends AppCompatActivity {
        Button mButton;
        ImageView mImageView;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                mImageView = (ImageView) findViewById(R.id.ImageView1);
                mButton = (Button) findViewById(R.id.Button1);
                mButton.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                downloadImage();
                        };
                });
        }
```

Thread Example (MainActivity.java)

```
public void downloadImage() {
        new Thread(new Runnable() {
                private Bitmap loadImageFromNetwork(String url) {
                        try {
                                Bitmap bitmap = BitmapFactory.decodeStream
                                ((InputStream)new URL(url).getContent());
                                return bitmap;
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                        return null;
                }
                public void run(){
                        final Bitmap bitmap= loadImageFromNetwork
                        ("https://i.cs.hku.hk/~twchim/c7506/android.png");
                        mImageView.setImageBitmap(bitmap);
                }
        }).start();
        }
}
```

The UI freezes or the program may even crash because you broke the rule "never update UI directly from worker thread"! At this moment, your main thread may be accessing the same UI view.

Thread Example 2 (Working)

With the same activity_main.xml and AndroidManifest.xml

Solution 1: View.post(Runnable)

```
public class MainActivity extends AppCompatActivity {
        Button mButton;
        ImageView mImageView;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                mImageView = (ImageView) findViewById(R.id.ImageView1);
                mButton = (Button) findViewById(R.id.Button1);
                mButton.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                downloadImage();
                        };
                });
        }
```

Solution 1: View.post(Runnable)

```
public void downloadImage() {
        new Thread(new Runnable() {
                private Bitmap loadImageFromNetwork(String url) {
                        try {
                                Bitmap bitmap = BitmapFactory.decodeStream
                                ((InputStream)new URL(url).getContent());
                                return bitmap;
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                        return null;
                }
                public void run(){
                        final Bitmap bitmap= loadImageFromNetwork
                        ("https://i.cs.hku.hk/~twchim/c7506/android.png");
                        mImageView.post(new Runnable(){
                                public void run() {
                                        mImageView.setImageBitmap(bitmap);
                                }
                        });
                }
        }).start();
    }
}
```

View.post(Runnable) makes the runnable join a queue. The runnable will be served on when the UI component is ready.

Last version:

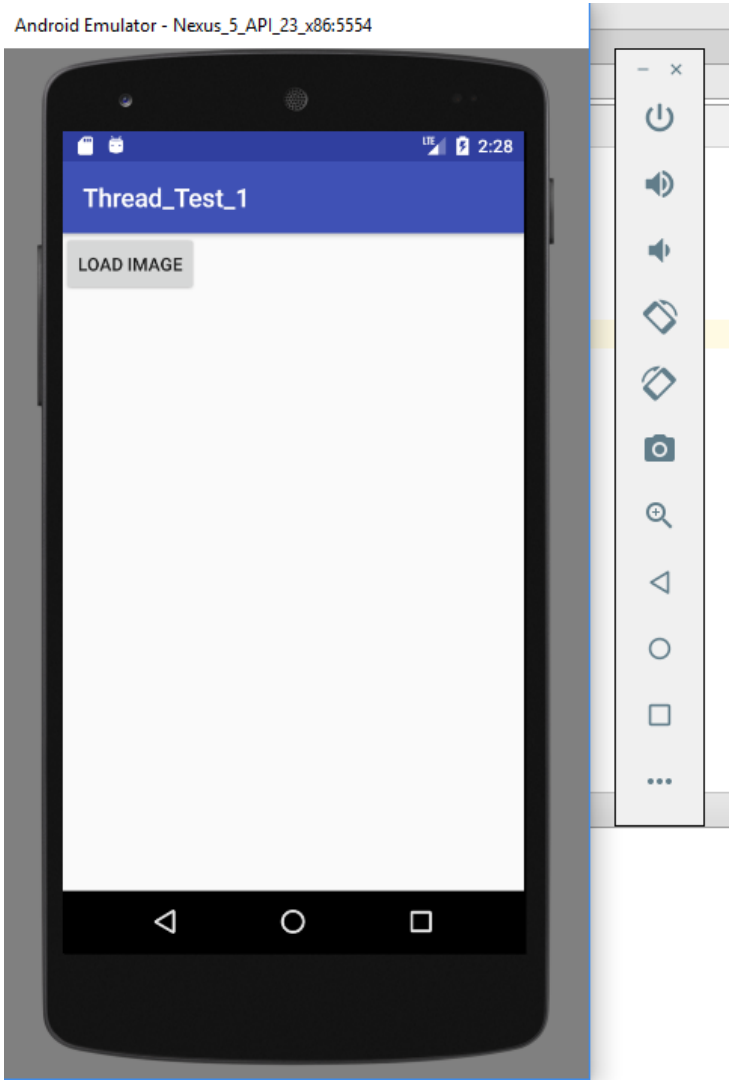
```
final Bitmap bitmap= …;
mImageView.setImageBitmap(bitmap);
```

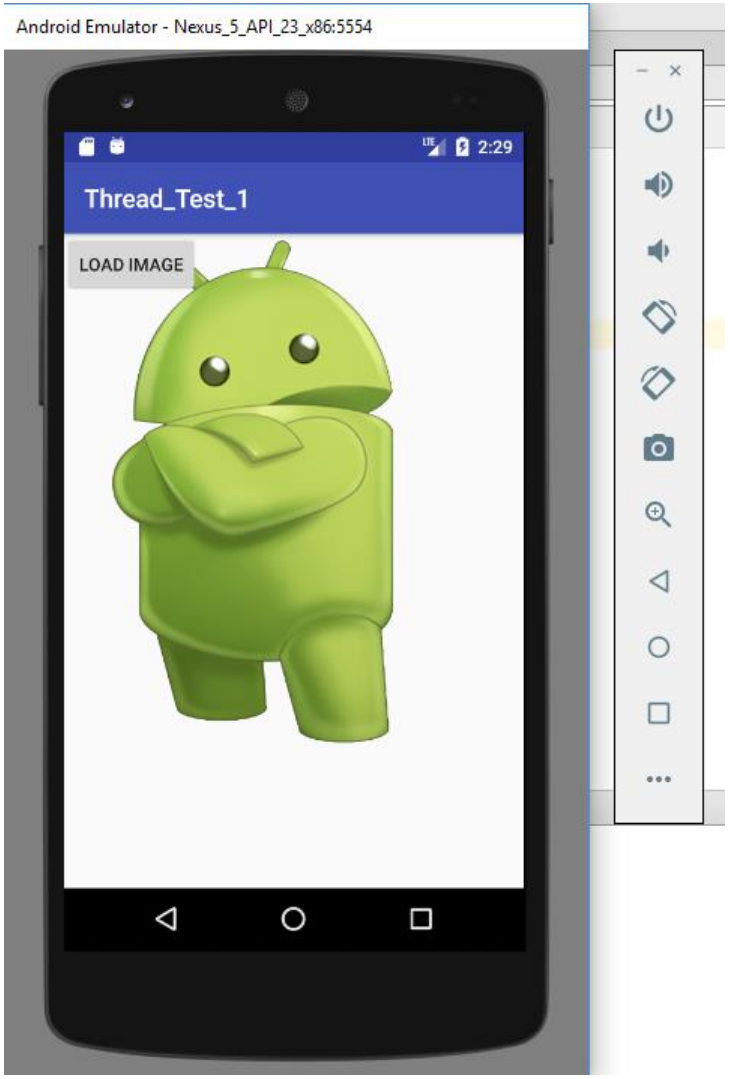
Solution 1: View.post(Runnable)

- post: Causes the Runnable to be added to the message queue
- Runnable: Represents a command that can be executed. Often used to run code in a different Thread.
- run(): Starts executing the active part of the class's code. This method is called when a thread is started and has been created with a class which implements Runnable.
- code: Operations on the UI view

- Implication: Threads which want to access an UI view will join a queue. Only one thread can access the UI view at any time.

Sample Output





Thread Example 3 (Working)

With the same activity_main.xml and AndroidManifest.xml

Solution 2: Executor

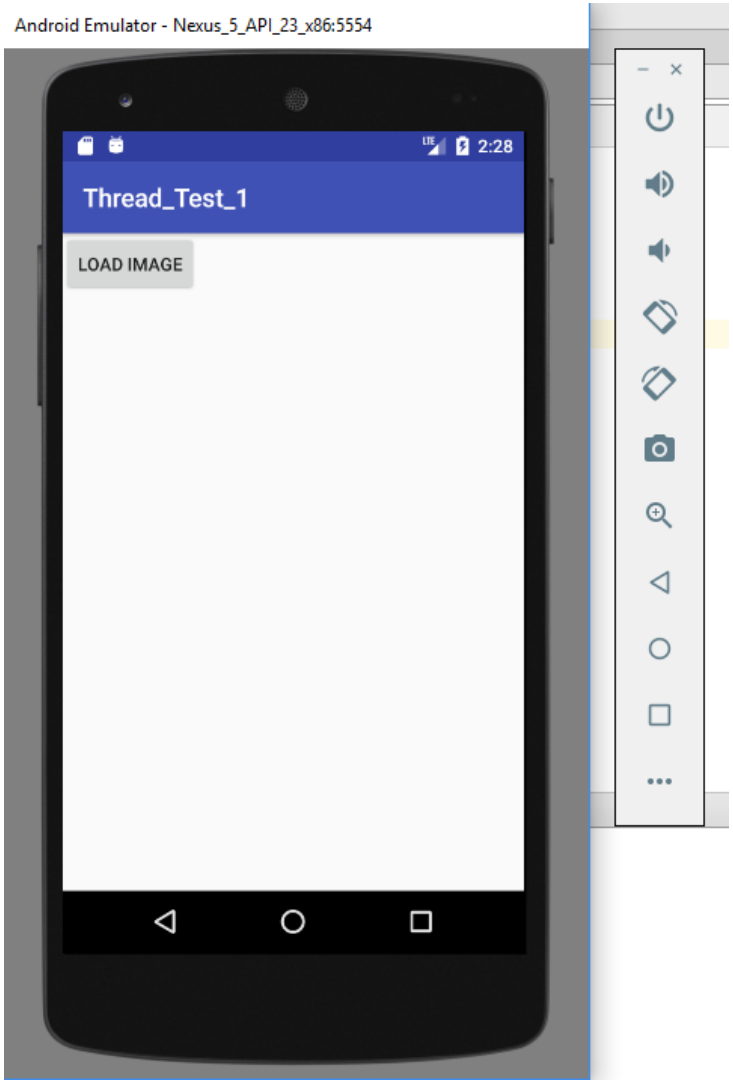
```
public class MainActivity extends AppCompatActivity {
        Button mButton;
        ImageView mImageView;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);
                mImageView = (ImageView) findViewById(R.id.ImageView1);
                mButton = (Button) findViewById(R.id.Button1);
                mButton.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                downloadImage();
                        };
                });
        }
```

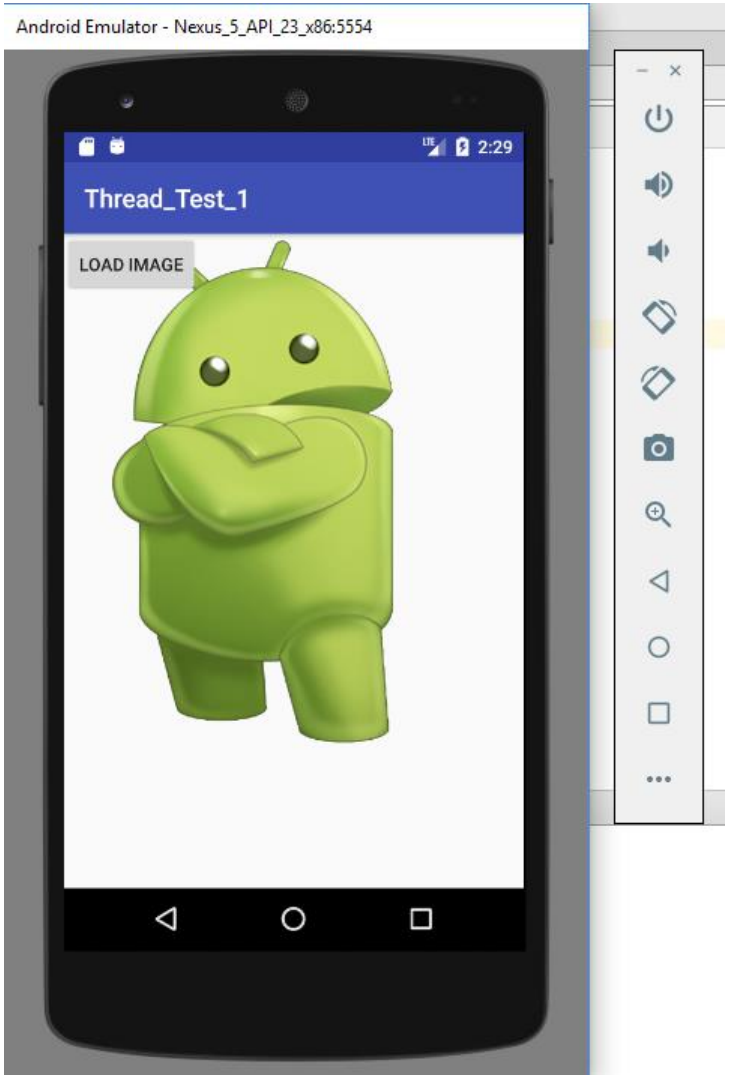
Solution 2: Executor

```
public void downloadImage() {
        ExecutorService executor = Executors.newSingleThreadExecutor();
        final Handler handler = new Handler(Looper.getMainLooper());
        executor.execute(new Runnable() {
                private Bitmap loadImageFromNetwork(String url) {
                        try {
                                Bitmap bitmap = BitmapFactory.decodeStream(
                                        (InputStream) new URL(url).getContent());
                                return bitmap;
                        } catch (Exception e) {   e.printStackTrace();   }
                        return null;
                }
                @Override
                public void run() {
                        final Bitmap bitmap = loadImageFromNetwork
                                ("https://i.cs.hku.hk/~twchim/c7506/android.png");
                        handler.post(new Runnable() {
                                @Override
                                public void run() {
                                        ImageView1.setImageBitmap(bitmap);
                                }
                        });
                }
        });
}
```

Note: Handlers can be used to communicate with UI thread (update the UI).

Sample Output





Stopping a Thread

Challenge: How to stop a thread?

- Statements in the run() method will be executed one by one. You are not supposed to stop the thread in the middle.
- However, if the run() method contains a loop, you can use a flag to terminate its repetitive execution.
- Please refer to the example on the next slide.

Challenge: How to stop a thread?

```
private boolean ThreadOn = true;
new Thread() {
        public void run() {
                …
                int i = 0;
                while (ThreadOn && i < 100) {
                        …
                }
        }
}.start();
…
```

At any point you want to terminate the while loop, you just need to set ThreadOn to false.

Handler

Challenge

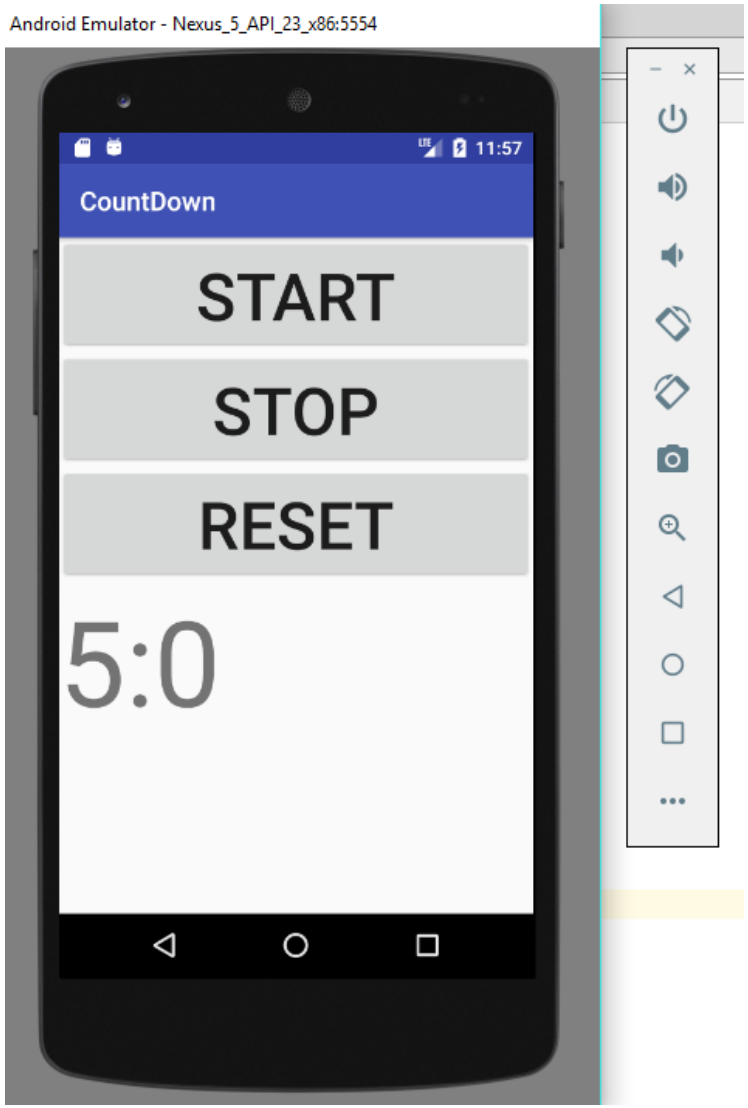
- Recall that we have to obey the rule “Do not access the UI toolkit from outside the UI thread”.
- Then how to implement a timer, in which the worker thread needs to update a TextView regularly?

Thread vs. Handler

- A runnable object can be started by a thread or a handler.
- Threads are generic processing tasks that can do most things, EXCEPT that they cannot update the UI.
- Handlers are bound to threads that allow you to communicate with the UI thread (update the UI).
- With handler you can also have things like scheduling and repeating.
- We often use handler to implement timer.

Countdown Timer

- Let's implement a countdown timer for our presentations.
- 3 Buttons and a TextView
  - "START" button: Start or continue the count down
  - "STOP" button: Stop the count down
  - "RESET" button: Reset the timer to 5:0



Syntax Issues

- Define a Handler object:

```
private Handler handler = new Handler();
```

- Define the runnable object:

```
private Runnable ABC = new Runnable() {
        public void run() {
                ...
        }
};
```

- Remove any pending posts of runnable ABC that are in the message queue.

```
handler.removeCallbacks(ABC);
```

- Make the runnable ABC to be added to the message queue, to be run after the specified amount of time elapses (in milliseconds). This has to be done at the beginning (for the first invocation) as well as at the end of the run() method (for the repetitive invocation).

```
handler.postDelayed(ABC, 1000);
```

Source Code (activity_main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="hkucs.countdown.MainActivity">
   <Button
      android:id="@+id/startButton"
      android:text="START"
      android:textSize="50sp"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
   <Button
      android:id="@+id/stopButton"
      android:text="STOP"
      android:textSize="50sp"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
   <Button
      android:id="@+id/resetButton"
      android:text="RESET"
      android:textSize="50sp"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
   <TextView
      android:id="@+id/timeDisplay"
      android:text="00:00"
      android:textSize="90sp"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
</LinearLayout>
```

Source Code (MainActivity.java)

```
package hkucs.countdown;

import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    static int maxTime = 300;
    int time;
    Button startButton, stopButton, resetButton;
    TextView timeDisplay;
    Boolean timerEnabled;
    private Handler handler = new Handler();
```

Source Code (MainActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    time = maxTime;
    timerEnabled = true;

    startButton = (Button) findViewById(R.id.startButton);
    stopButton = (Button) findViewById(R.id.stopButton);
    resetButton = (Button) findViewById(R.id.resetButton);
    timeDisplay = (TextView) findViewById(R.id.timeDisplay);

    int Minutes = time / 60;
    int Seconds = time % 60;
    timeDisplay.setText(Minutes + ":" + Seconds);
```

Source Code (MainActivity.java)

```
    startButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            handler.removeCallbacks(updateTimer);
            handler.postDelayed(updateTimer, 1000);
            timerEnabled = true;
        }
    });
    stopButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            timerEnabled = false;
        }
    });
    resetButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            time = maxTime;
            int Minutes = time / 60;
            int Seconds = time % 60;
            timeDisplay.setText(Minutes + ":" + Seconds);
        }
    });
}
```

Source Code (MainActivity.java)

```
    private Runnable updateTimer = new Runnable() {
        public void run() {
            time = time - 1;
            int Minutes = time / 60;
            int Seconds = time % 60;
            timeDisplay.setText(Minutes + ":" + Seconds);
            if ((timerEnabled) && (time > 0)) {
                handler.postDelayed(this, 1000);
            }
        }
    };
}
```

"timerEnabled" variable:
Trick to stop a repeating runnable!

iOS

Grand Central Dispatch (GCD)

- Grand Central Dispatch was created by Apple over 10 years ago to help developers write multi-threaded code without manually creating and managing the threads.
- With GCD, Apple took an asynchronous design approach to the problem. Instead of creating threads directly, we use GCD to schedule work tasks, and the system will perform these tasks by making the best use of its resources.
- GCD will handle creating the requisite threads and will schedule tasks on those threads, shifting the burden of thread management from the developer to the system.
- A big advantage of GCD is that you don't have to worry about hardware resources as we write our code. GCD manages a thread pool, and it will scale from a single-core Apple Watch all the way up to a many-core MacBook Pro.

Dispatch Queues

- Main building blocks of GCD, which let us execute arbitrary blocks of code using a set of parameters defined.
- Tasks in dispatch queues are always started in a first-in, first-out (FIFO) fashion. But because the completion time of tasks depends on several factors, and is not guaranteed to be FIFO.
- Serial vs. Concurrent:
  - A serial queue performs only one task at the time.
  - A concurrent queue allows us to execute multiple tasks at the same time.
- 3 kinds of queues available:
  - Main dispatch queue (serial, pre-defined)
  - Global queues (concurrent, pre-defined)
  - Private queues (can be serial or concurrent, created by developer)

Main Queue & Concurrent Queues

- Every app comes with a Main queue, which is a serial queue that executes tasks on the main thread.
- This queue is responsible for drawing application's UI and responding to user interactions (touch, scroll, etc.).
- If you block this queue for too long, your iOS app will appear to freeze.
- When performing a long-running task (network call, computationally intensive work, etc.), we avoid freezing the UI by performing this work on a background queue. Then we update the UI with the results on the main queue.
- In addition to the main queue, every app comes with several pre-defined concurrent queues that have varying levels of Quality of Service (an abstract notion of priority in GCD.)

Sync vs. Async

- When we dispatch a task to a queue, we can choose to do so synchronously or asynchronously using the sync and async dispatch functions.
- **Sync:** When our code reaches a sync statement, it will block the current queue (Main dispatch queue if we call our sync statement inside viewDidLoad) until that task completes. Once the task returns/completes, control is returned to the caller, and the code that follows the sync task will continue.
- **Async:** An async statement, on the other hand, will execute asynchronously with respect to the current queue, and immediately returns control back to the caller without waiting for the contents of the async closure to execute. There is no guarantee as to when exactly the code inside that async closure will execute.

Examples

- Submit work asynchronously to the user interactive (highest priority) QoS queue:

```
DispatchQueue.global(qos: .userInteractive).async {
        …
        print("global concurrent queue!")
}
```

- Call the default priority global queue by not specifying a QoS:

```
DispatchQueue.global().async {
        …
        print(“global generic queue")
}
```

- Create our own private serial queue (serial by default):

```
let serial = DispatchQueue(label: “hk.hku.cs.serial-queue")
serial.sync {
        …
        print(“private serial queue")
}
```

- Create our own private concurrent queue:

```
let concurrent = DispatchQueue(label: “hk.hku.cs.serial-queue“, attributes: .concurrent)
concurrent.async {
        …
        print(“private concurrent queue")
}
```

More details: https://developer.apple.com/documentation/dispatch/dispatchqueue

Chapter 8.

End

2023-2024
COMP7506 Smart Phone Apps Development
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong