COMP4521 EMBEDDED SYSTEMS SOFTWARE

LAB 4: WRITING MULTI-THREADED APPLICATIONS

INTRODUCTION

In this lab, we will see the problems encountered in doing long-running processing on the main UI thread. We then examine two different approaches to create a multi-threaded applications: using threads/handlers and using AsyncTask, both of which permit long-running processing to be done in the background, while at the same time permit the UI to be updated regularly on the progress.

OBJECTIVES

- Learn to use progress bars.
- Learn to create new threads/handlers.
- Learn to use AsyncTask.

A SIMPLE EXAMPLE SHOWING THE PROBLEMS OF LONG-RUNNING PROCESSING IN THE MAIN UI THREAD

1. Locate the SDK of Android for Eclipse. In Eclipse folder, run eclipse.bat, set your workspace to D:\Temp, click **Window**-> **Preferences**-> **Android**, and choose the SDK location (must be D:\comp4521\android-sdk-windows) as where you have put the Android SDK.
2. Create an Android Virtual Device (AVD).
3. Create a new Android project. The project name, application name and activity name is *Unthreaded*, Target is Android 4.0.3, package name is *hkust.course.comp4521*.
4. Modify the main layout. Open **/res/layout/main.xml**, and use following code as its new content:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
```

```
    <TextView
        android:id="@+id/status_text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:paddingTop="20dip"
        android:text="Click the button" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="left"
        android:paddingTop="20dip"
        android:text="Progress:" />

    <ProgressBar
        android:id="@+id/progress_bar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="fill_parent"
        android:layout_height="40dip"
        android:indeterminate="false"
        android:padding="10dip" />

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:gravity="center_horizontal"
        android:orientation="horizontal" >

        <Button
            android:id="@+id/start_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="20dip"
            android:padding="14dip"
            android:text="Start" />

        <Button
            android:id="@+id/reset_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="20dip"
            android:padding="14dip"
            android:text="Reset" />
    </LinearLayout>

</LinearLayout>
```

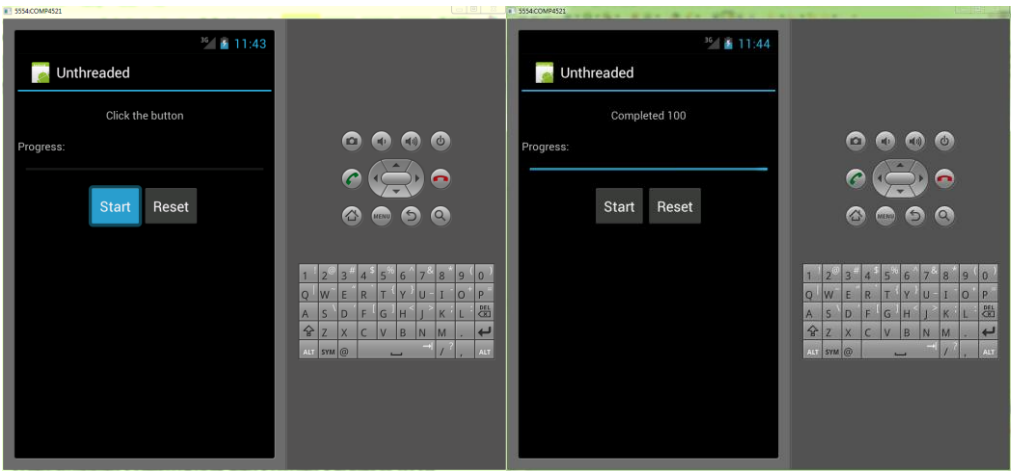
Notice how progress bar and buttons are defined in the layout above.

5. Modify the source code of the main activity as per the source code of the Unthreaded class given below. Read the comments carefully to check how long-running processing is being done in the OnClick() listener of the Start button.

```
public class main extends Activity implements OnClickListener{

     private ProgressBar progressBar;

     private TextView statusText;

     private int completed;

     /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // Get handles for the progress bar and status text TextView
        progressBar = (ProgressBar) findViewById(R.id.progress_bar);
        statusText = (TextView) findViewById(R.id.status_text);

        // Set the maximum value that the progress bar will display
        progressBar.setMax(100);

        // Declare the listeners for the two buttons
        Button startButton = (Button) findViewById(R.id.start_button);
        startButton.setOnClickListener(this);
        Button resetButton = (Button) findViewById(R.id.reset_button);
        resetButton.setOnClickListener(this);
    }

     public void onClick(View source) {
          // Start button is clicked
          if (source.getId() == R.id.start_button) {
               int l;

               // Initialize the progress bar and the status TextView
               completed = 0;
               progressBar.setProgress(completed);
               statusText.setText(String.format("Completed %d", completed));

               // Do a large amount of computation
               for (int i = 0; i < 100; ++i) {
                    for (int j = 0; j < 5000; ++j) {
                         for (int k = 0; k < 5000; ++k) {
                              l = i*j*k;
                         }
                    }

                    // Periodically update the UI to reflect the progress of the computation
                    completed += 1;
                    progressBar.setProgress(completed);
                    statusText.setText(String.format("Completed %d", completed));
               }
          }
          // Reset button is clicked
          else if (source.getId() == R.id.reset_button) {
               progressBar.setProgress(0);
               statusText.setText(String.format("Click the button"));
          }
     }
}
```

6. Run the application and observe the behavior of the application. This leads to an unresponsive UI, freezing the UI and potentially leading to the Application Not Responding (ANR) dialog popping up. The progress bar will not show the progress correctly; instead, it will directly jump to 100% when the job is done. This is because the

job is done in the only thread of the application, which is responsible for updating the UI. When it's busy doing the calculation, it cannot update the UI concurrently.



CREATE A BACKGROUND THREAD FOR DOING THE LONG-RUNNING COMPUTATION AND POSTING PROGRESS TO UI USING HANDLER

1. Modify the previous Unthreaded.java. First, declare a new variable of type handler in the Unthreaded class:

```
private Handler handler;
```

2. Define a handler in the *OnCreate()* method to handle messages from the background thread which do the computing.

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

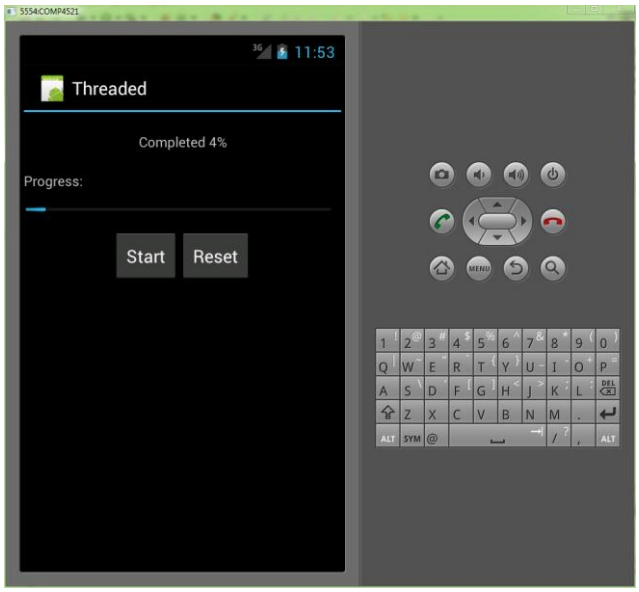
3. Modify the event of the start button. Recall that in our previous version, clicking on the start button will trigger a loop. Now we modify the event so that a new thread which executes the loop will be called rather than executing the loop inside the main thread.

```
// Declare a new background thread
Thread workerthread = new Thread(worker);

// Start the background thread
workerthread.start();
```

4. Notice that a new thread is created using the runnable object *worker*. We expect this worker to do the long-running processing in the background, and inform the main UI thread on the progress periodically. Here we only give part of the source code. Please read the comments and try to finish the code by yourself.

```
private Runnable worker = new Runnable() {

     public void run() {

          int l;

          // Initialize the progress bar and the status TextView
          completed = 0;

          // we want to modify the progress bar so we need to do it from the UI thread
          // how can we make sure the code runs in the UI thread? use the handler!

          handler.post(new Runnable() {
               public void run() {
                    // Here the worker thread uses the post method to initialize the
progress bar.
                    // Finish this method by yourself.

               }
          });

          // Here the worker thread do the loop and update the progress bar periodically
          // Similarly, it uses the handler.post method to update the progress bar
          // Finish this part by yourself.

};
```

5. Run the application. Now you can see the progress bar working properly. You can change the name of the application to “Threaded”.



AN EASIER WAY OF DOING BACKGROUND PROCESSING:

ASYNCTASK

1. Android offers a class, called the AsyncTask, to do background processing. When using AsyncTask, we no longer need to declare handlers. Instead, declare a new variable *worker* of type *WorkerTask*. We will define *WorkerTask* later. And delete the declaration of *handler* in the *OnCreate()* method.

```
WorkerTask worker;
```

2. Modify the events of *OnClick()*. Since we use AsyncTask instead of creating new threads by ourselves, it is different to create a new AsyncTask.

```
// Declare a new Async Task of the WorkerTask class
worker = new WorkerTask();

// Execute the Async Task
worker.execute();
```

3. Create the *WorkerTask* class by extending AsyncTask. Here we only give part of the source code. Please read the comments and try to finish the code by yourself.

```
private class WorkerTask extends AsyncTask<Object, String, Boolean> {


    // Initialize the progress bar and the status TextView
   @Override
   protected void onPreExecute() {

           completed = 0;

           // This will result in a call to onProgressUpdate()
           publishProgress();
   }

   @Override
  // This method updates the main UI, refreshing the progress bar and TextView.
  // Finish this method by yourself.
   protected void onProgressUpdate(String... values) {

   }

   // Do the main computation in the background and update the UI using publishProgress()
   // Finish this method by yourself.
   @Override
      protected Boolean doInBackground(Object... params) {


           }
           return null;
      }
```

4. Run the application. The result will be similar to the use of background thread, but with simpler code.