가천대학교
Gachon University

학생 여러분 반갑습니다.

다른 친구들이 입장할 때까지 조금 기다려 주십시오.

곧 모바일 프로그래밍 수업을 시작합니다.

음소거( )가 되었는지 확인 바랍니다.

모바일 프로그래밍
화목(1,2교시)/ 화목(3,4교시)
정윤현 (AI/소프트웨어학부)

Mobile Programming

Android Programming

Chap 7. Thread

Prof. Younhyun Jung

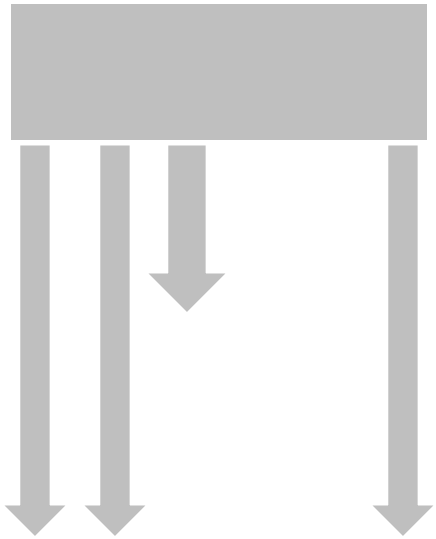
Email) younhyun.jung@gachon.ac.kr

가천대학교 소프트웨어학과
Gachon University

Threads

1. A Thread is
   - a flow of control in a process
   - a **concurrent** unit of execution.

2. Each virtual machine instance has at least one **main thread** .

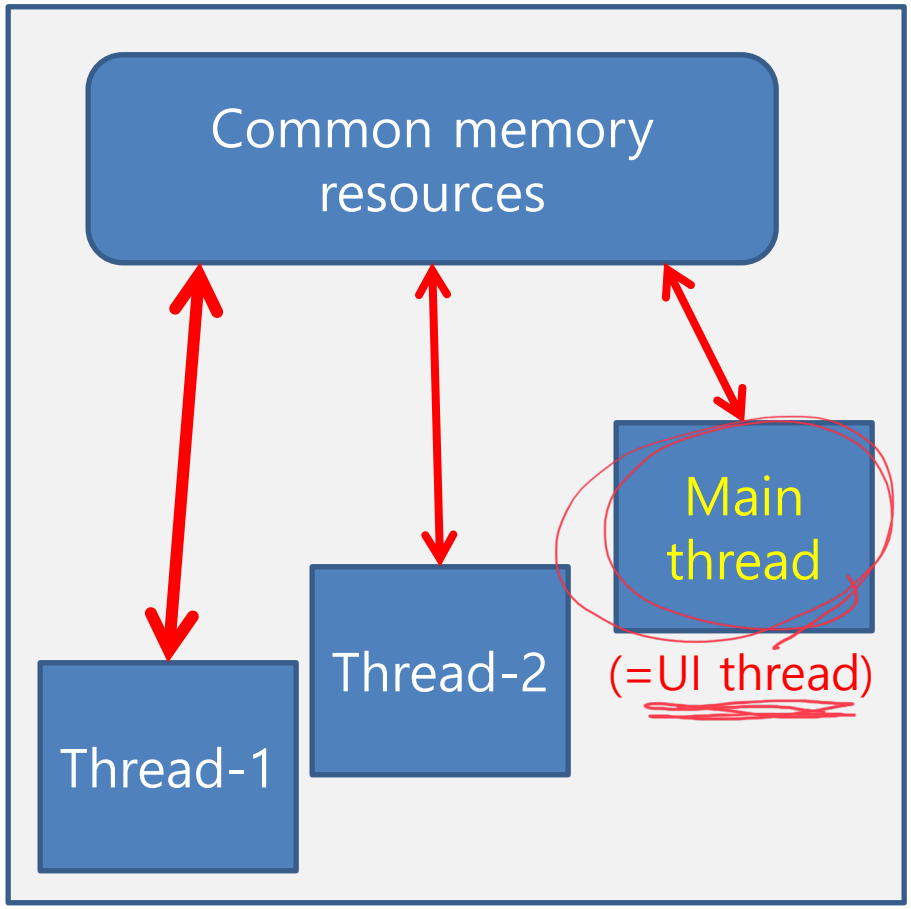


- The application might decide to launch additional Threads for specific purposes.

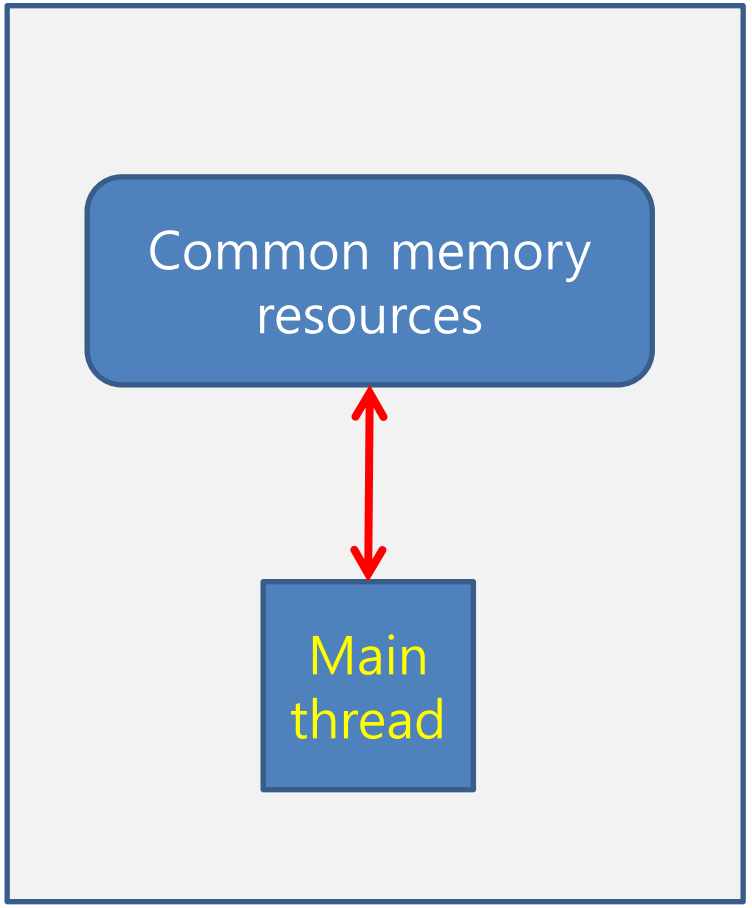
http://developer.android.com/guide/components/processes-and-threads.html

Threads

Process 1 (Dalvik Virtual Machine 1)



Process 2 (Dalvik Virtual Machine 2)



Threads

- There are basically two main ways of having a **Thread** execution application code.
  - Create a new class that *extends* **Thread** and override its **run()** method.

```
MyThread t = new MyThread();

t.start();
```

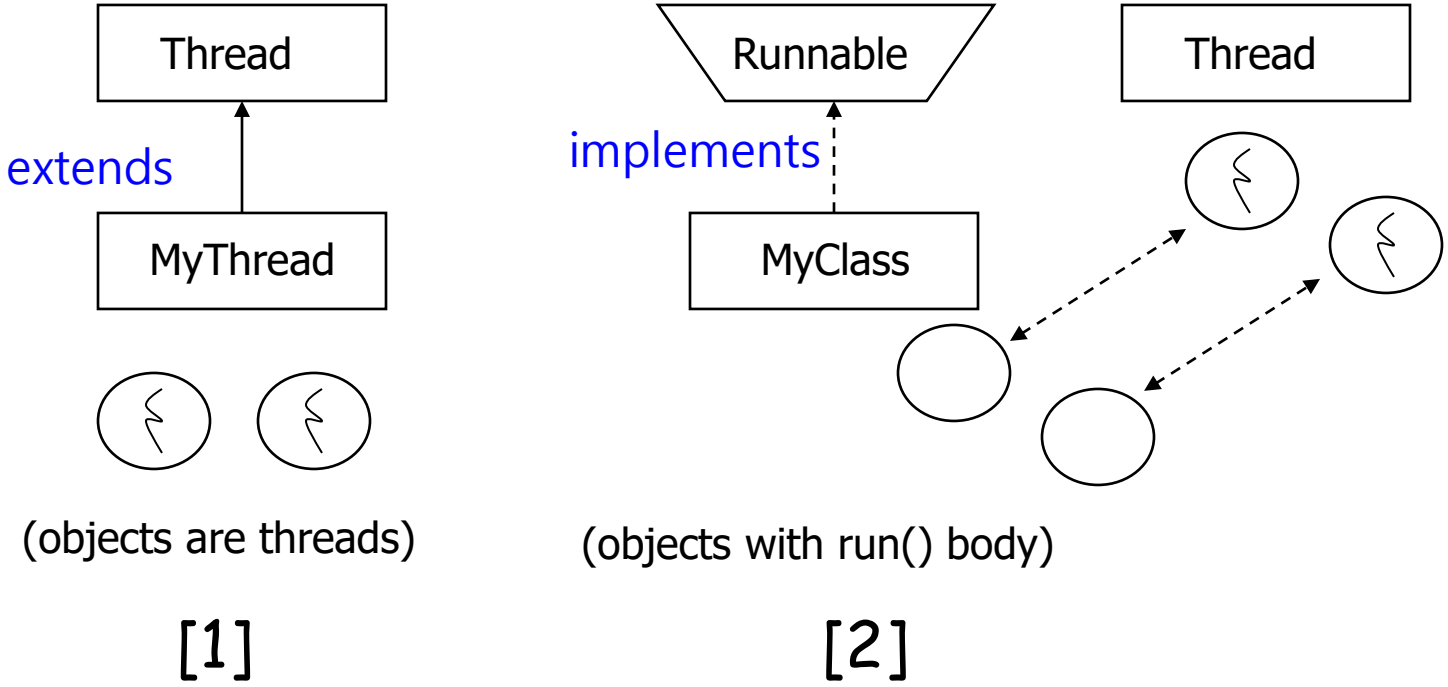
  - Create a new **Thread** instance passing to it a **Runnable** object.

```
Runnable myRunnable1 = new MyRunnableClass();

Thread t1 = new Thread(myRunnable1);

t1.start();
```

In both cases, the start() method must be called to actually execute the new Thread.

Two ways to use Thread

- Create a class that extends the **`Thread class`**
- Create a class that implements the **`Runnable`** interface



1st method: Extending Thread class

- Create a class by extending Thread class and override run() method:

```
class MyThread extends Thread
{
     public void run()
            {
       // thread body of execution
            }
}
```

- **Create a thread:**

```
MyThread thr1 = new MyThread();
```

- **Start Execution of threads:**

```
thr1.start();
```

- or Create and Execute together:

```
new MyThread().start();
```

2nd method: Implementing Runnable interface

- Create a class that implements the interface Runnable and override run() method:

```
class Tasks implements Runnable
{
   .....
   public void run()
   {
      // thread body of execution
   }
}
```

- **Creating Object:**

```
Tasks myObject = new Tasks();
```

- **Creating Thread Object:**

```
Thread thr1 = new Thread( myObject );
```

- **Start Execution:**

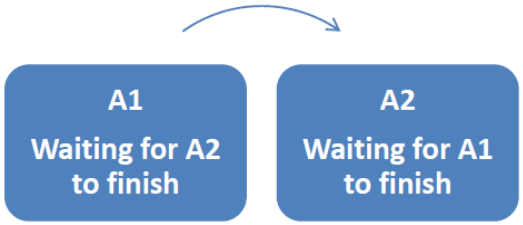
```
thr1.start();
```

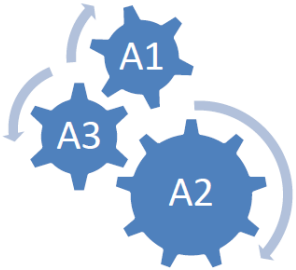
Why Multi-threading?

- Main advantages
  - Threads share the process' resources but are able to execute independently.
  - Application responsibilities can be separated
    - main thread runs UI, and
    - slow tasks are sent to background threads
  - Threading provides an useful abstraction of concurrent execution.
  - A multithreaded program operates faster on computer systems that have multiple CPUs.
- Disadvantages
  - Code tends to be more complex
  - Need to detect, avoid, resolve deadlocks

→ message passing 등
↳ 안드가 구현 지원 되어 있어서..

- So,
  - Threads in the same VM interact with and synchronize by the use of shared objects and monitors associated with these objects.
  - Need Concurrency control





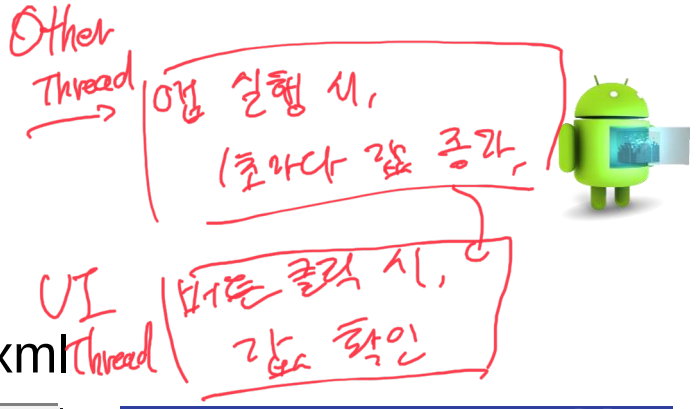
Two Simple Rules in using Threads

- The Android UI toolkit is not thread-safe. So, you must not manipulate your UI from a worker thread—you must do all manipulation to your user interface from the UI thread.

- there are simply two rules to Android's single thread model:

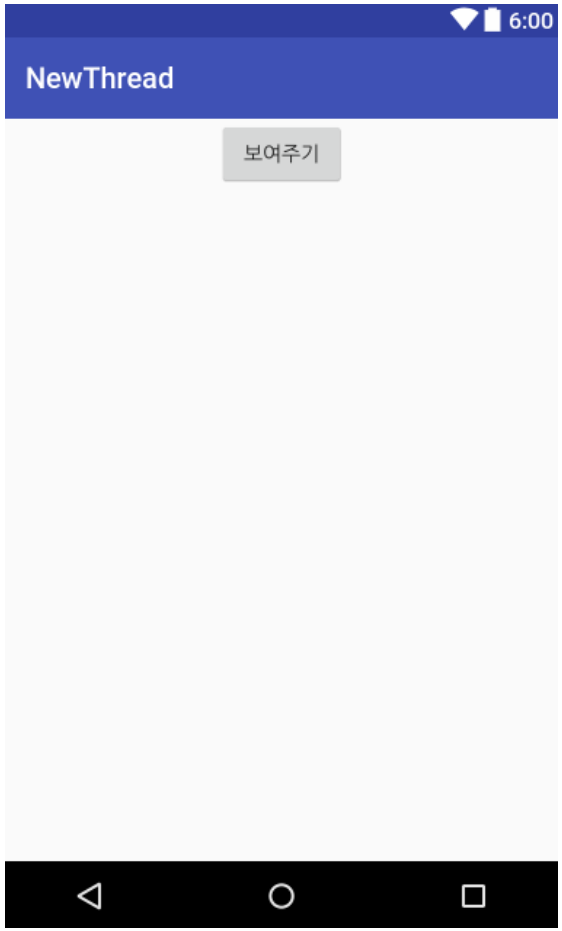
1. Do not block the UI thread
2. Do not access the Android UI toolkit from outside the UI thread

Exercise



- Using Thread at Android – activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
   >

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="보여주기"
        android:layout_centerHorizontal="true"/>
    <TextView
        android:id="@+id/textView01"
        android:layout_width="match_parent"
        android:layout_height="300dp"
        android:layout_below="@+id/button"
        android:layout_centerHorizontal="true"/>
</RelativeLayout>
```



Exercise

- Using Thread at Android – MainActivity.java

```
package org.androidtown.listview.newthread;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private boolean running;
    private int value;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final TextView textview = (TextView) findViewById(R.id.textView01);
        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                textview.setText("스레드에서 받은 값: " + value);
            }
        });
    }
```

Exercise

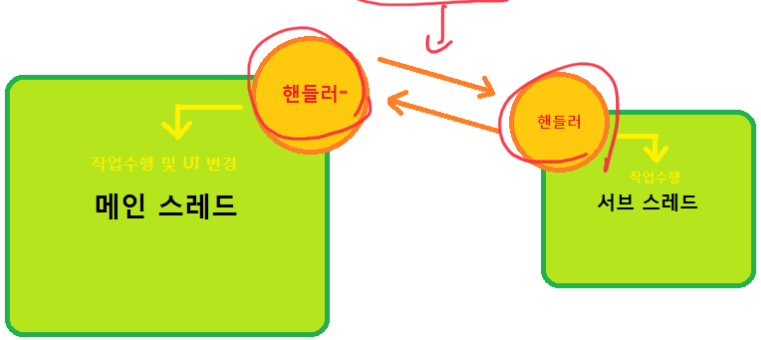
- Using Thread at Android – MainActivity.java

```
@Override
    protected void onResume() {
        super.onResume();

        running = true;
        Thread thread1 = new BackgroundThread();
        thread1.start();
    }

    @Override
    protected void onPause() {
        super.onPause();
        running = false;
        value = 0;
    }

class BackgroundThread extends Thread {
    public void run(){
        while (running){
            try{Thread.sleep(1000);
            value++;
            }catch(Exception ex){}
        }
    }
}}
```

sleep()에 의해 일시 정지 상태가 된 thread는 지정된 시간이 다되거나 interrupt발생 시 항상 InterupptedException가 자동발생되어 잠에서 깨어나 실행대기상태가 됨. 따라서, 의무적으로 예외처리를 해줘야 정상실행가능

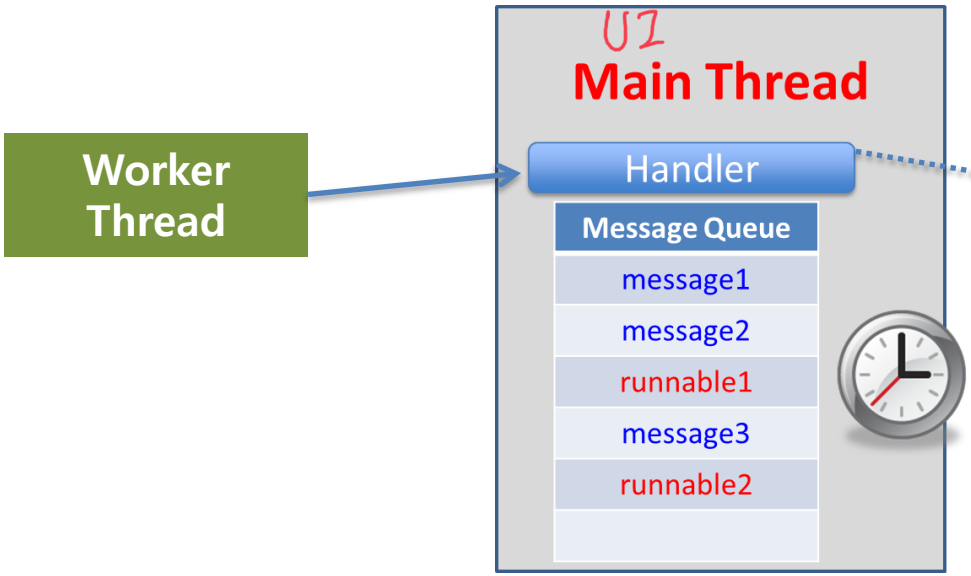
Communication with Handler and Message-Queue

- Inter-Thread Communications via Handler & Message queue
  - Handler
    - A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue.



Interaction between Android threads (Main and background) is accomplished using (a)a main thread ***Handler*** object and (b)posting ***Runnable*** **objects**/ sending **Messages** to the main thread.

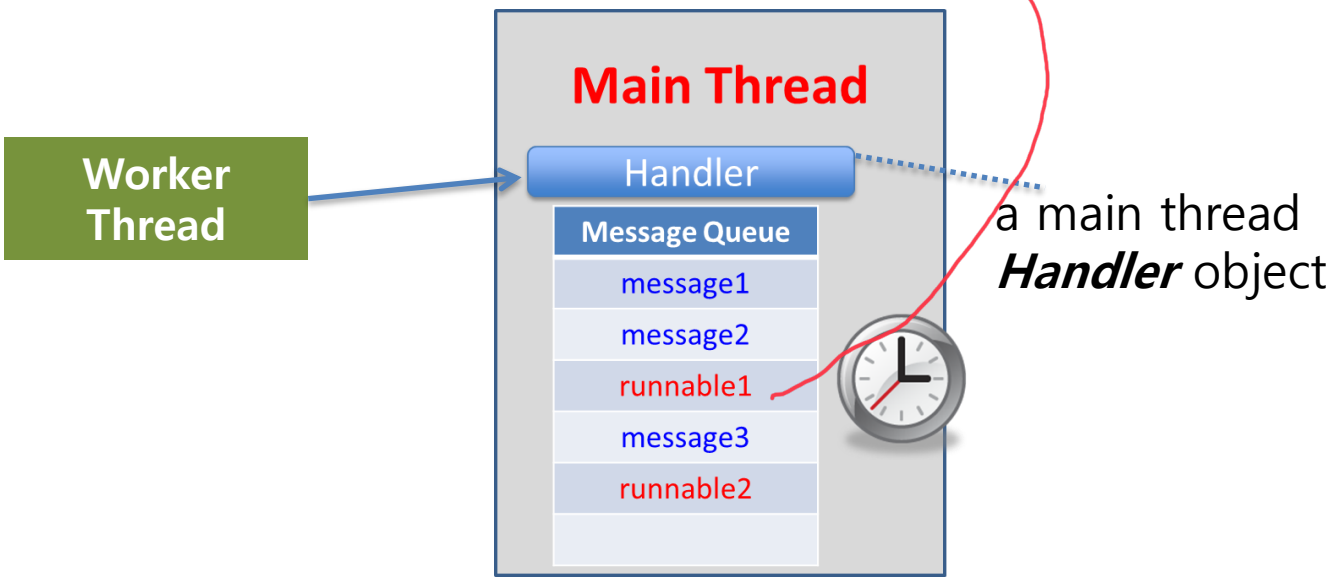
Communication with Handler and Message-Queue




```
public class MainActivity extends Activity {

// handler for the main thread
 Handler handler = new Handler() {
  @Override
  public void handleMessage(Message msg) {

  //display each item in a single line

}

};
```

Inter-Thread Communications

- What types of data does a Worker thread send to the Main thread?
  - Contents (message) → **Messages**
  - Routines (executable codes) → **Runnable objects**



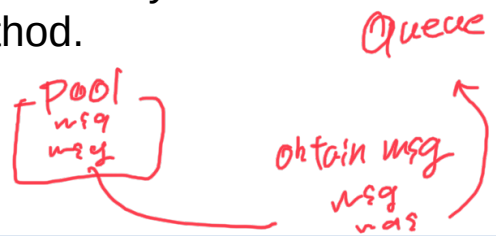
Handler Class

- Handler
  - A Handler allows you to send and process Message and Runnable objects associated with a thread's MessageQueue.
    - 다른 객체들이 보낸 메시지를 받고 처리하는 객체
  - You can create a new Handler in a thread, then the Handler is bound to the message queue of the thread in which it is created.
    - So, you have to create a Handler in the UI thread in order to access the UI component.
  - Then, the handler can be accessed by other threads (=worker threads), and the handler is used to communicate with the thread.
    - The Handler will deliver messages or runnables to that message queue and execute them as they come out of the message queue.
    - Hander object는 연결된 쓰레드로 부터 오는 message나 runnable object를 받으면 그를 처리해주는 call back 함수가 불려지고 해당 message/runnable을 처리하는 코드를 실행시키는 식으로 운영된다. (runnable ; 실행시키고자하는 루틴!)

http://developer.android.com/reference/android/os/Handler.html

Thread interaction: Messages

- A secondary thread that wants to communicate with the main thread.
- To send a Message to a Handler of the main thread, the secondary thread must request a message token using the ***obtainMessage*()** method.
  - to get the Message object out of the pool.
  - Queue 상에 값을 채워 넣을 토큰을 얻어온다고 생각하자



- **Example**
  - final int MSG_ID1 = 1
  - // thread 1 produces some local data

```
String localData = "Greeting from thread 1";
```

  - // thread 1 requests a message & **fill** localData to it

```
Message msg = myHandler.obtainMessage (MSG_ID1, localData);
// 이렇게 채운 msg는 sendMessage로 전달!
```

- Message obtainMessage()
- Message obtainMessage(int what)
- Message obtainMessage(int what, Object obj)
- Message obtainMessage(int what, int arg1, int arg2)
- Message obtainMessage(int what, int arg1, int arg2, Object obj)

- Once obtained, the background thread can fill data into the message token and attach it to the Handler's message queue using the ***sendMessage*()** method or request the execution of runnable objects through the ***post*()** method.

Processing Messages

Then,

- the Handler uses the *handleMessage*() method to continuously take care of new messages arriving to the main thread.
  - To process messages sent by the background threads, your Handler needs to implement the listener
    (note: other sub-threads also can define their own handler objects, so they can implement handleMessage methods.)

    handleMessage(. . .)

    which will be called with each message that appears on the message queue.

- The handler can update the UI as needed. However, it should still do that work quickly, as other UI work is suspended until the Handler is done.

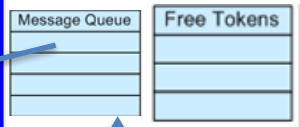
Step for Thread interaction

onCreate에 넣장-

```
public class MainActivity extends Activity {

// handler for the main thread
 Handler mHandler = new Handler() {
  @Override
  public void handleMessage(Message msg) {
   String returnedValue =(String)msg.obj;
   txt.setText(returnValue);
      }
 };
```

Handler

MAIN THREAD



.obtainMessage()

what

.sendMessage(msg)


```
Message msg = mHandler.obtainMessage(1,
(String)data);

mHandler.sendMessage(msg);
```

Worker Thread
(Background thread)

Using sendMessages

Main Thread

```
...
Handler myHandler = new Handler() {

  @Override
  public void handleMessage(Message msg) {

     // do something with the message...
     // update GUI if needed!
     ...
  }//handleMessage

};//myHandler

...
```

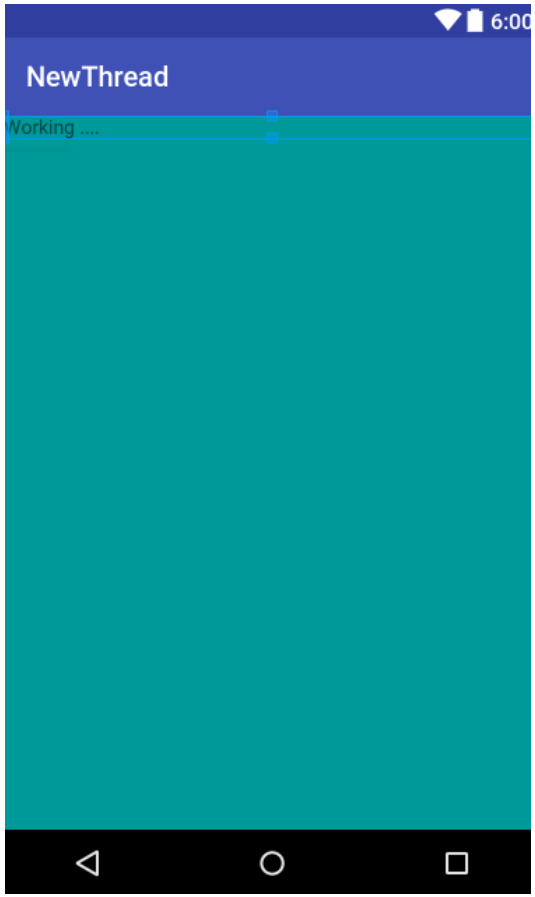
Background Thread

```
...
Thread backgJob = new Thread (new Runnable (){

  @Override
  public void run() {
   //...do some busy work here ...
   //get a token to be added to
    //the main's message queue
   Message msg = myHandler.obtainMessage();
     ...
    //deliver message to the
    //main's message-queue
   myHandler.sendMessage(msg);
  }//run

});//Thread

//this call executes the parallel thread
backgroundJob.start();
...
```

Exercise

- Using Handler to access main thread
  - Make a layout !

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/widget28"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#ff009999"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android"
    >
    <TextView
        android:id="@+id/TextView01"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Working ...."/>
    <ProgressBar
        android:id="@+id/progress"
        style="?android:attr/progressBarStyleHorizontal"
        android:max="100"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```



Exercise

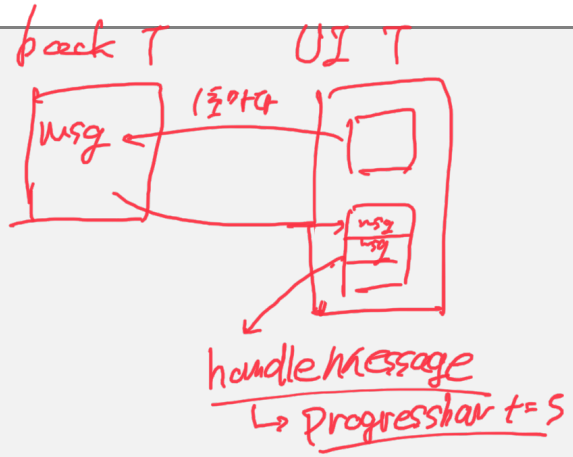
- MainActivity.java

```
package org.androidtown.listview.newthread;

import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    TextView textView0;
    ProgressBar bar;
    ProgressHandler handler;
    boolean isRunning = false;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView0 = (TextView) findViewById(R.id.TextView01);
        bar = (ProgressBar) findViewById(R.id.progress);

        handler = new ProgressHandler();

    }
```

Exercise

- MainActivity.java

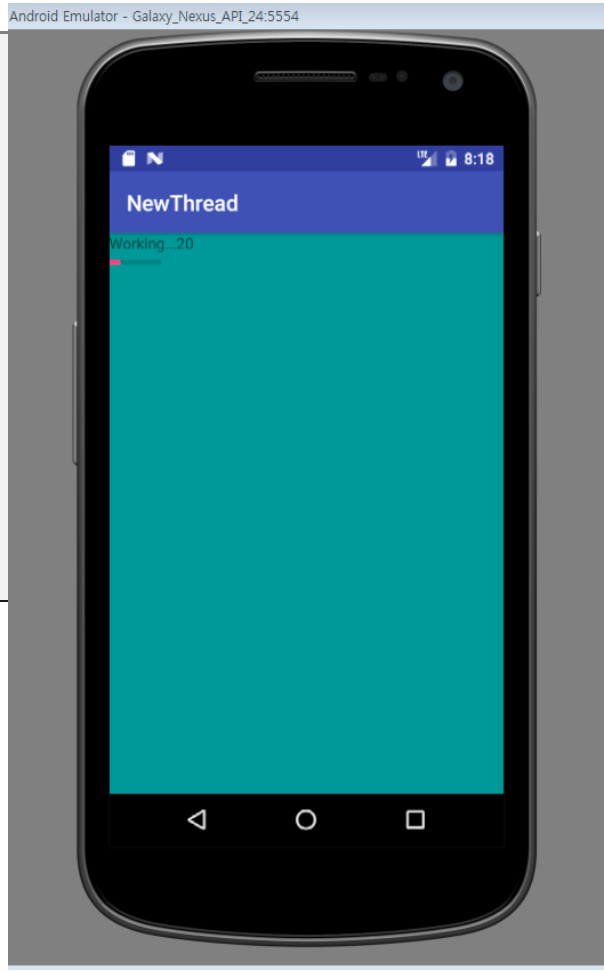
```
@Override
protected void onStart() {
    super.onStart();
    bar.setProgress(0);
    Thread thread1 = new Thread(new Runnable(){
        public void run(){
            try{for(int i=0; i<20 && isRunning; i++){
            Thread.sleep(1000);
            Message msg = handler.obtainMessage();
            handler.sendMessage(msg);

            }}
        catch(Exception ex){
            Log.e("MainActivity", "Exception in processing message.", ex);
        }}
        });
    isRunning = true;
    thread1.start();
}

@Override
protected void onStop() {
    super.onStop();
    isRunning = false;
}
```



Exercise

- MainActivity.java

```
public class ProgressHandler extends Handler{
        public void handleMessage(Message msg){
            bar.incrementProgressBy(5);
            if(bar.getProgress()==bar.getMax()){
                textView0.setText("Done");
            } else {
                textView0.setText("Working..."+bar.getProgress());
            }

        }
    }

}
```



- RUN!!

Exercise

- Other form to conduct same job!
  - Modify previous MainActivity.java

```
package org.androidtown.listview.newthread2;

import android.os.Bundle;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    TextView textView0;
    ProgressBar bar;
    Handler handler;
    ProgressRunnable runnable;
    boolean isRunning = false;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textView0 = (TextView) findViewById(R.id.TextView01);
        bar = (ProgressBar) findViewById(R.id.progress);

        handler = new Handler();
        runnable = new ProgressRunnable();

    }
```

Exercise

- Other form to conduct same job!
  - Modify previous MainActivity.java

```
@Override
    protected void onStart() {
        super.onStart();
        bar.setProgress(0);
        Thread thread1 = new Thread(new Runnable(){
            public void run(){
                try{for(int i=0; i<20 && isRunning; i++){
                    Thread.sleep(1000);
                    handler.post(runnable);

                }}
                catch(Exception ex){
                    Log.e("MainActivity", "Exception in processing message.", ex);
                }}
        });
        isRunning = true;
        thread1.start();
    }

    @Override
    protected void onStop() {
        super.onStop();
        isRunning = false;
    }
```

Handler의 post() method는
runnable객체내의 run() 메소드를
실행하게 함

Exercise

- Other form to conduct same job!
  - Modify previous MainActivity.java

```
public class ProgressRunnable implements Runnable {
            public void run(){
            bar.incrementProgressBy(5);

            if(bar.getProgress()==bar.getMax()){
                textView0.setText("Done");
            } else {
                textView0.setText("Working..."+bar.getProgress());
            }

        }
    }

}
```

AsyncTask class

- **AsyncTask** enables proper and easy use of the UI thread.
  - Widely used for short operations (a few seconds at the most) such as downloading, login and so on.
    - 다운로드나 로그인, 등등 UI와 로직이 같이 실행되야 되는 등의 구현에 용이! (널리 활용되고 있음)
- **AsyncTask** class allows to perform background operations and publish results on the UI thread **without having to manipulate threads and/or handlers.**
- An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread.

AsyncTask class (cont.)

- An asynchronous task is defined by
  - 3 generic types, called Params, Progress and Result,
  - 4 steps, called onPreExecute(), doInBackground(), onProgressUpdate() and onPostExecute(),
  - 1 auxiliary (보조) method, publishProgress()

3 Generic Types	4 Main States	1 Auxiliary Method
Params, Progress, Result	onPreExecute, doInBackground, onProgressUpdate onPostExecute.	publishProgress

4 Steps for an AsyncTask

- **onPreExecute()**
  - invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
- **doInBackground()**
  - Invoked after onPreExecute() finishes executing
  - contains the coding instruction which should be performed in **a background thread.** This method runs automatically in a separate Thread.
- **onPostExecute()**
  - synchronize itself again with the user interface thread and allows to update it. This method is called by the framework once the doInBackground() method finishes

4 Steps for an AsyncTask

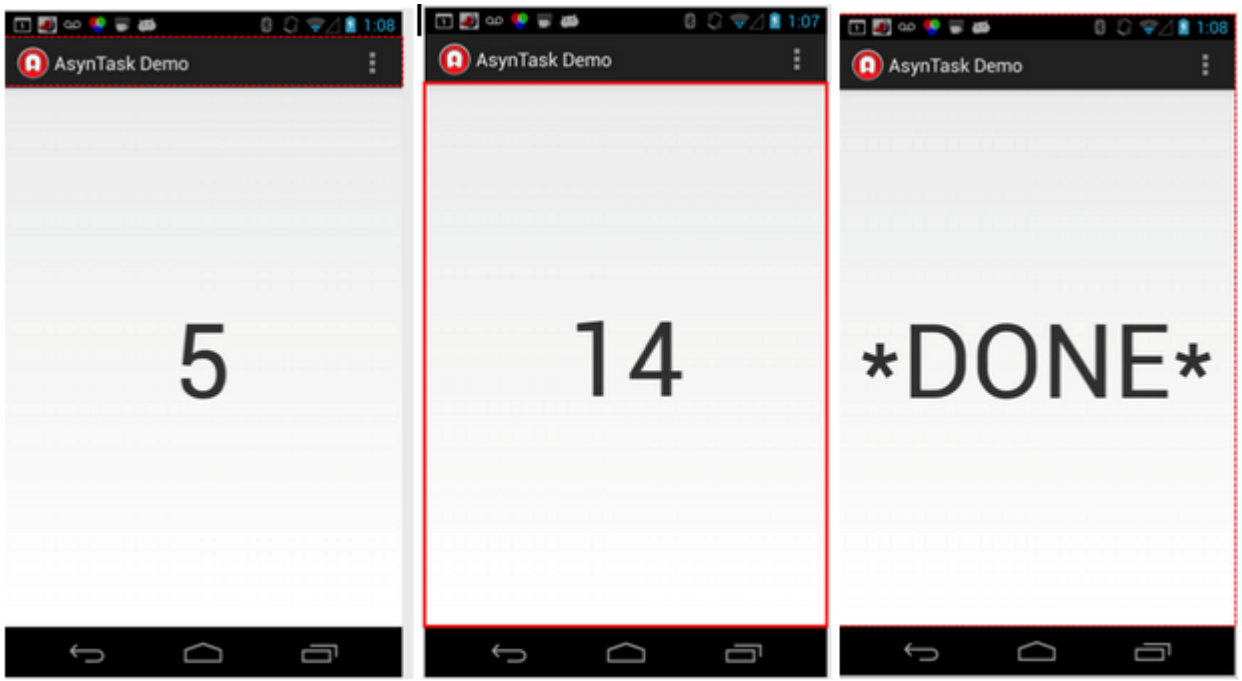
- **onProgressUpdate(Progress...)**
  - invoked on the UI thread after a call to publishProgress(Progress...) from onInBackground() method
  - Executed in the UI thread. This method is used to display any form of progress in the user interface while the background computation is still executing.

AsyncTask's methods

- onPreExecute( )
  - doInBackground작업이 시작되기 전에 호출되며 UI 스레드(main thread)에서 실행된다.
- doInBackground (Params... params)
  - 배경 작업을 수행하며 분리된 스레드에서 실행된다 (Thread라 생각하면 됨)
- onProgressUpdate (Progress... values)
  - doInBackground에서 넘긴 values 값을 받아서 처리하는 부분
  - (doInBackground에서 publishProgress 메서드를 호출할 때 작업 경과 표시를 위해 호출되며 UI 스레드에서 실행된다.)
- onPostExecute (Result result) : 백그라운드 작업이 끝난 후 UI 스레드에서 실행된다.

Exercise

- New Project : AsyncTaskDemo
  - starts a AsyncTask to first count down from 15. Different text is rewritten out to the UI.



Exercise

- Layout – activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/widget28"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >


    <TextView
        android:id="@+id/tv_counter"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:gravity="center"
        android:textSize="90sp"/>

</RelativeLayout>
```

Exercise

```
package org.androidtown.listview.thread_review;

import android.os.AsyncTask;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    TextView tvCounter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tvCounter = (TextView) findViewById(R.id.tv_counter);

        new CountDownTask().execute(); //Starts the CountDownTask
    }
```

Exercise

```
private class CountDownTask extends AsyncTask<Void, Integer, Void>{

    @Override
    protected void onPreExecute() {
        tvCounter.setText("*START*");
    }

    @Override
    protected Void doInBackground(Void... params) {
        for(int i=15; i>=0; i--){
            try{Thread.sleep(1000);
            publishProgress(i); //Invokes onProgressUpdate();
            }catch (Exception e){}
        }
        return null;
    }

    @Override
    protected void onProgressUpdate(Integer... values) {
        tvCounter.setText(Integer.toString(values[0].intValue()));

    }

    @Override
    protected void onPostExecute(Void aVoid) {
        tvCounter.setText("*DONE*");        }
}

}
```