Multi-Threading in Android

Mobile App Programming

What we learn today?

- Let's make an application using multiple threads.
  - Learn about structure of android threads
    - **UI thread**
    - Background thread
  - Using **Handler** to send message or runnable instance

Concurrent vs Parallel

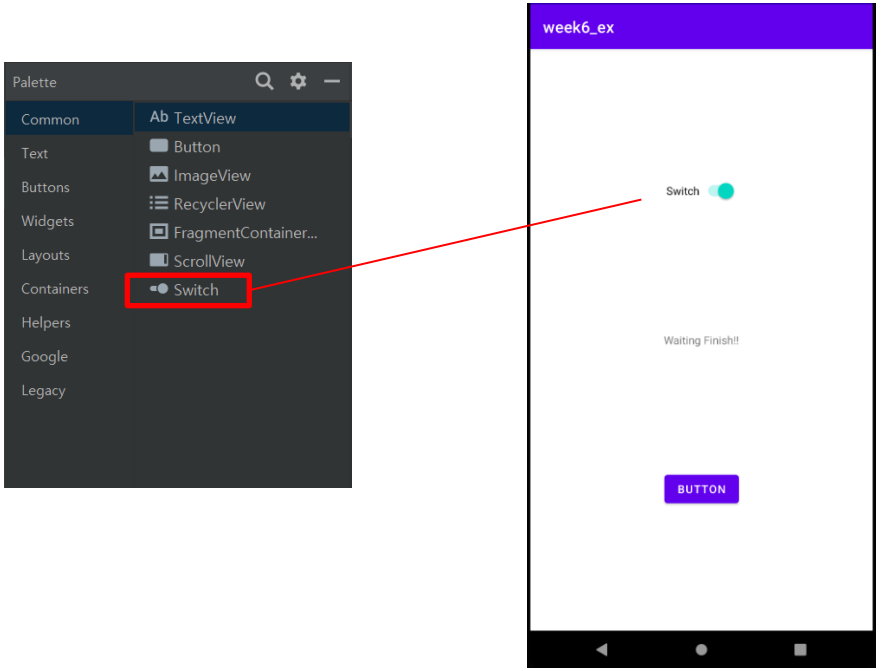
- **Concurrency** (동시성)
  - Concurrency relates to an application that is processing **more than one task at the same time**. Concurrency is an approach that is used for decreasing the response time of the system by using the **single processing unit**
- **Parallelism** (병렬성)
  - Parallelism is related to an application where tasks are **divided into smaller sub-tasks that are processed seemingly simultaneously or parallel**. It is used to increase the throughput and computational speed of the system by using **multiple processors.**

Concurrent Programming : Multi-Thread

- Most application run all components(Activity, Service, Broadcast Receiver, etc.) in the **same Thread**.
- This thread is called the **Main thread** (or **UI Thread**)
  - Android separate **UI thread** and **background thread**.
- Why we need threading in android?
  - Only UI thread can change UI screen.
  - If we run heavy workload on UI thread, UI tasks will be stuck.
  - Then, your application will not response to click event!

Example 1

- What is the problem of below application?

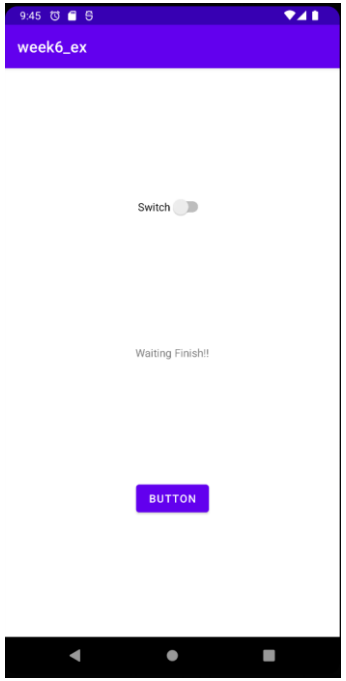



```java
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView = findViewById(R.id.textView);
        Button btn = findViewById(R.id.button);
        btn.setOnClickListener(view -> {
            synchronized (this){
                try {
                    wait(5000);
                    textView.setText("Waiting Finish!!");
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

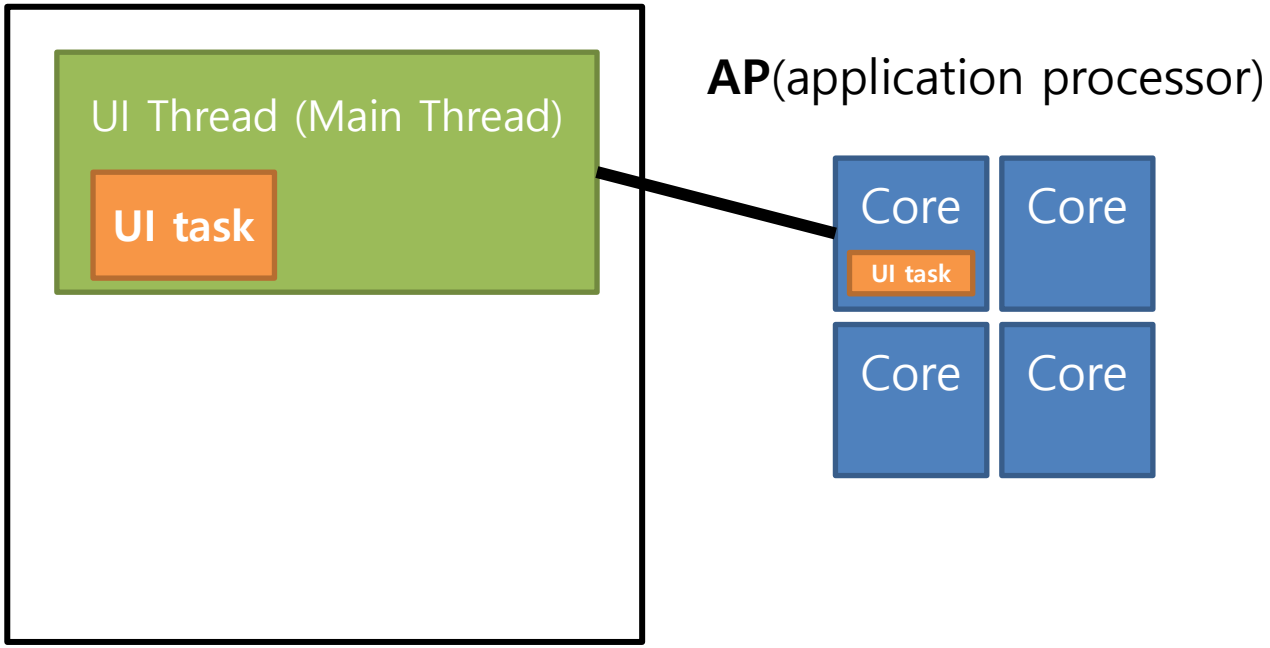
- UI(switch) doesn't work while waiting 5 seconds.
  (waiting 5 seconds means heavy workload)

Example 1) Problems

1) Click switch!

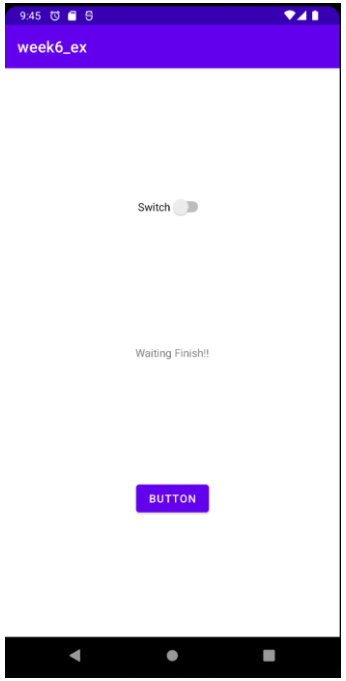


Your application

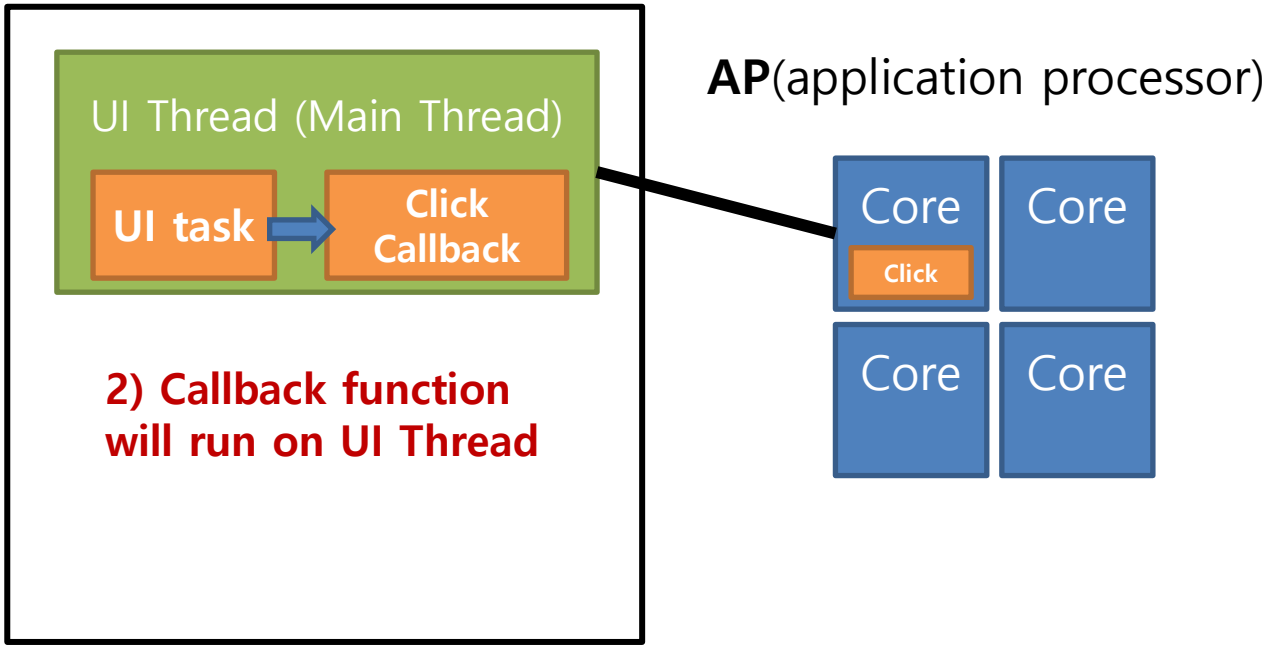


Example 1) Problems

1) Click switch!

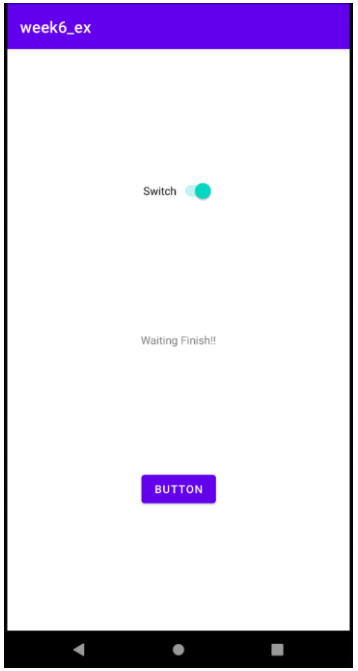


Your application



Example 1) Problems

1) Click switch!



Your application

UI Thread (Main Thread)

UI task

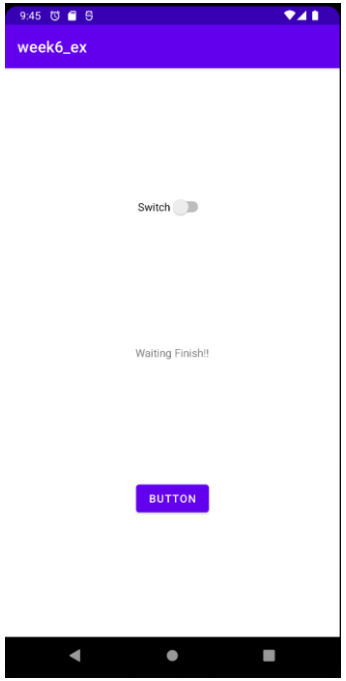
2) Callback function will run on UI Thread

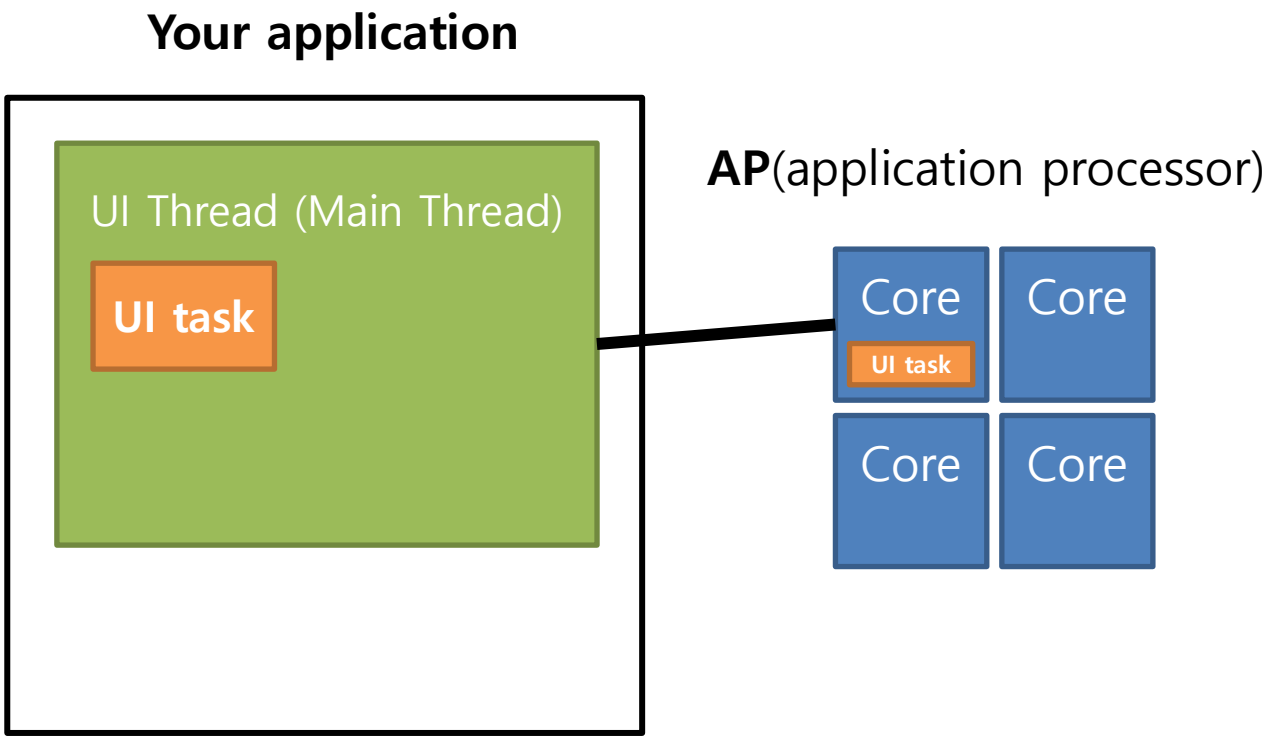
3) After handling callback UI task wait next event.

AP(application processor)

Example 1) Problems

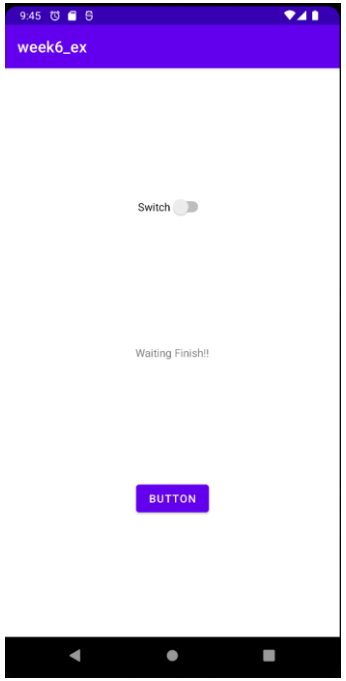
1) Click button!





Example 1) Problems

1) Click button!



Your application

UI Thread (Main Thread)

UI task → **Callback (wait 5 sec)**

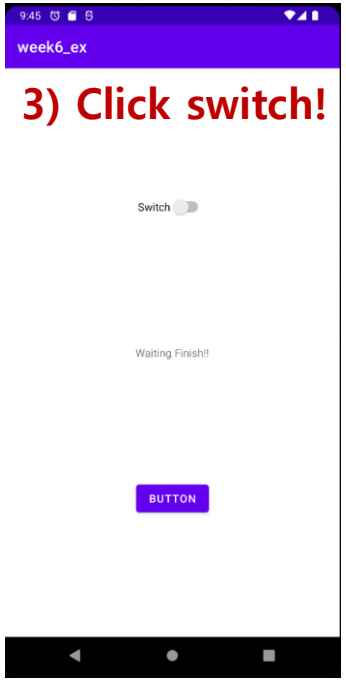
2) Wait 5 seconds(with Synchronized) in UI thread

AP(application processor)

Core | Core
Wait 5
Core | Core

Example 1) Problems

1) Click button!



3) Click switch!

Your application

UI Thread (Main Thread)

UI task → Callback (wait 5 sec)

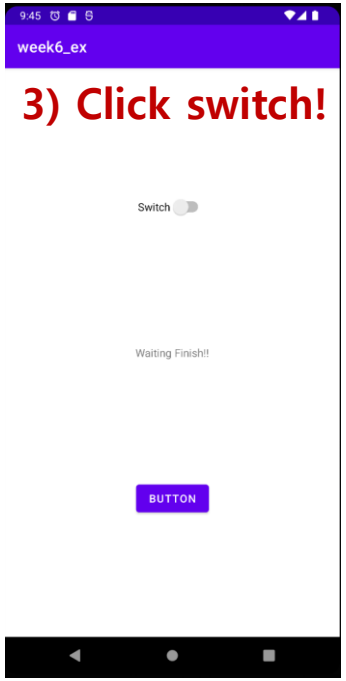
2) Wait 5 seconds(with Synchronized) in UI thread

AP(application processor)

Core (Wait 5) | Core | Core | Core

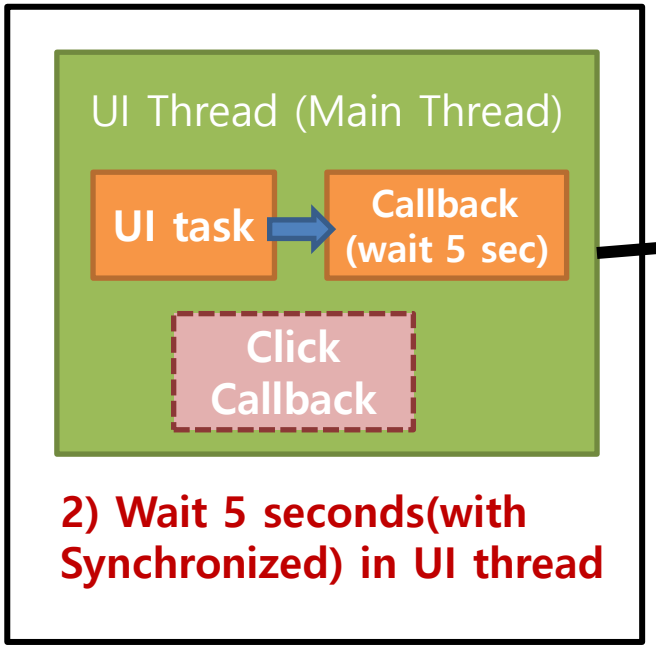
Example 1) Problems

1) Click button!



3) Click switch!

Your application



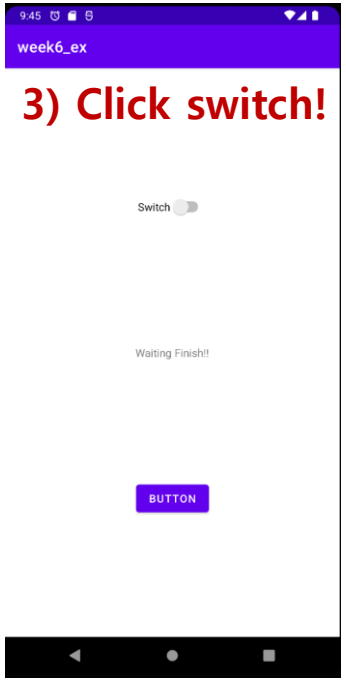
2) Wait 5 seconds(with Synchronized) in UI thread

AP(application processor)

4) Click callback can't run!

Example 1) Problems

1) Click button!



3) Click switch!

Your application

UI Thread (Main Thread)

UI task

Click Callback

2) Wait 5 seconds(with Synchronized) in UI thread

4) Click callback can't run!

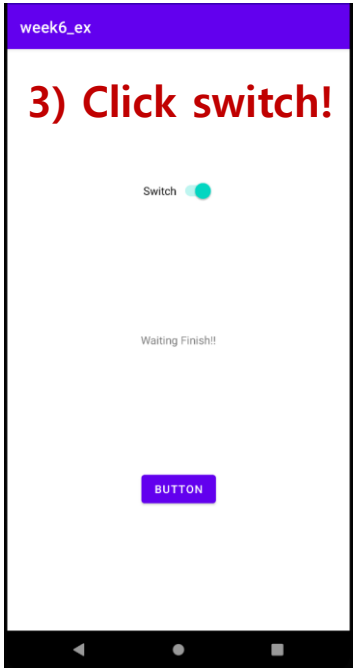
AP(application processor)

Core | Core
UI task
Core | Core

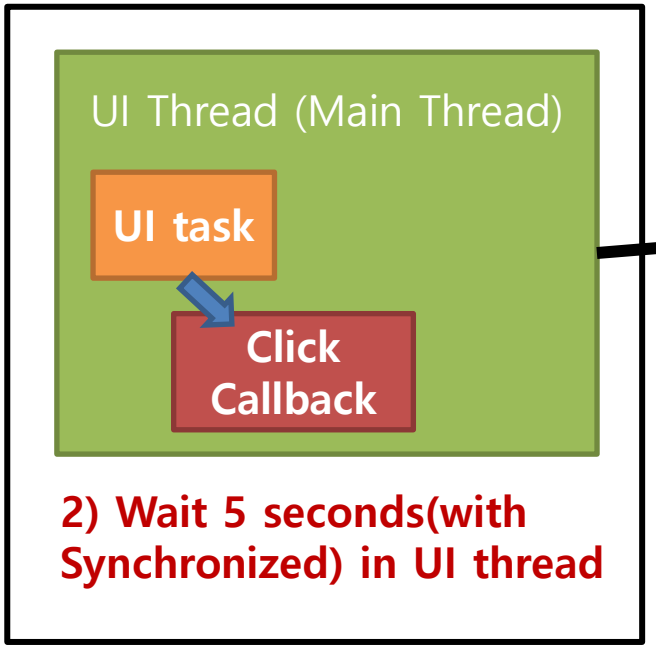
5) After 5 seconds ...

Example 1) Problems

1) Click button!



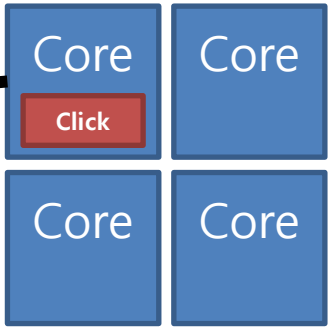
Your application



2) Wait 5 seconds(with Synchronized) in UI thread

4) Click callback can't run!

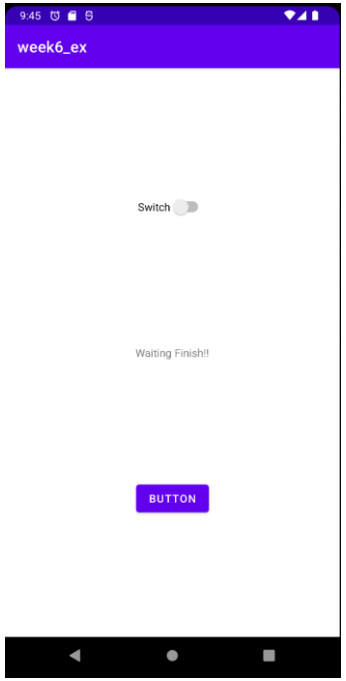
AP(application processor)

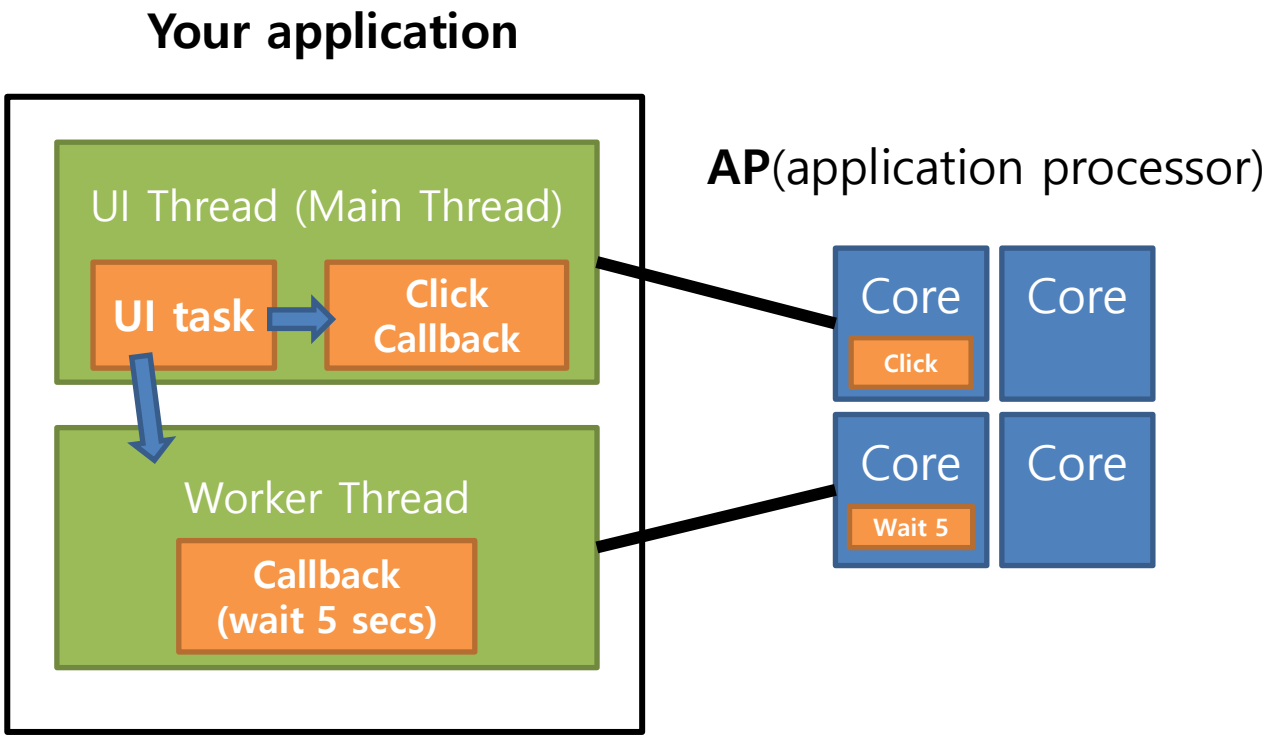


5) After 5 seconds ...

6) Run click callback

Example 1) Solution

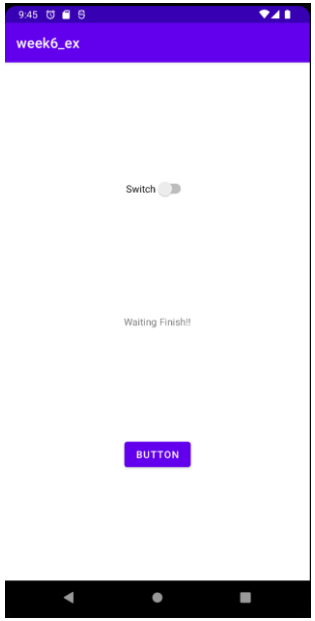


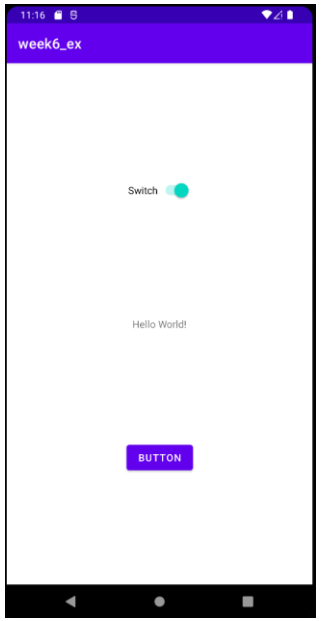


*** We must run heavy workload tasks(networking, File I/O ,,,) in worker thread.**

Example 1) Solution

Now, switch is clickable while waiting 5 seconds.





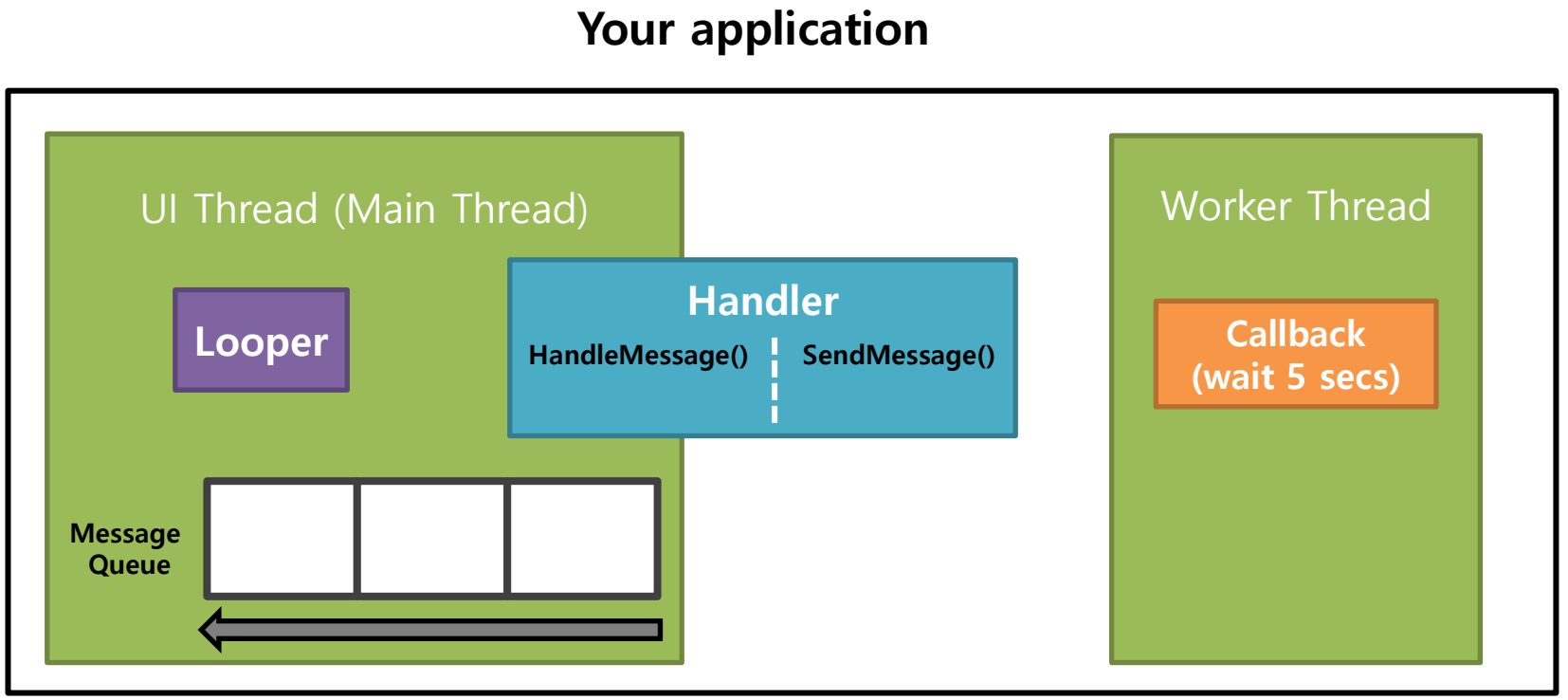

```java
public class MainActivity extends AppCompatActivity {

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

        TextView textView = findViewById(R.id.textView);
        Runnable runnable = new Runnable(){
            @Override
            public void run() {
                synchronized (this){
                    try {
                        wait(5000);  // kinds of heavy workload
                        textView.setText("Waiting Finish!");
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            }
        };
        Button btn = findViewById(R.id.button);
        btn.setOnClickListener(view -> {
            new Thread(runnable).start();
        });
    }
}
```

```
2022-03-29 15:16:46.214 14979-15021/? E/AndroidRuntime: FATAL EXCEPTION: Thread-2
    Process: com.example.week6_test, PID: 14979
    android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
```

*** But worker thread can't change UI => We should send msg to UI thread!!!**

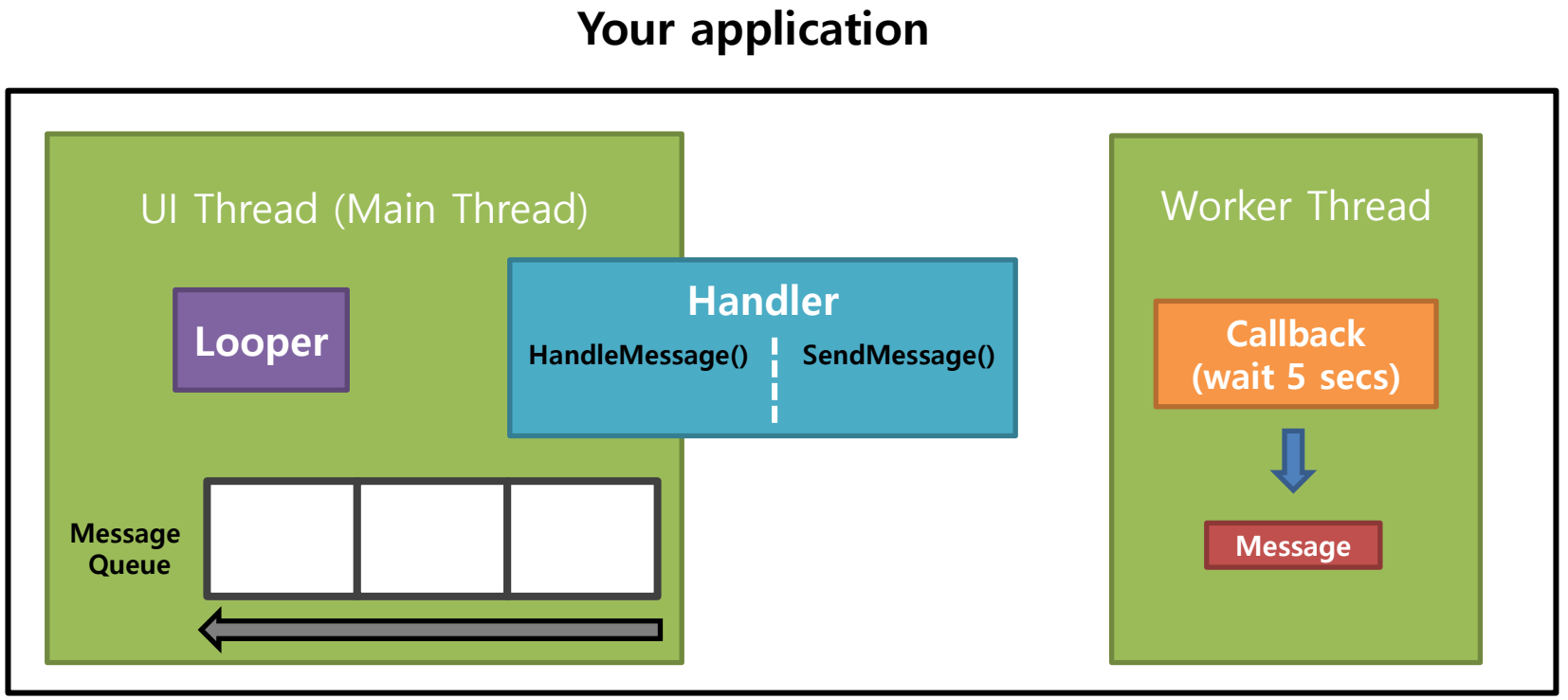
Example 2) Handler and Looper

- Using Handler you can send message or runnable instance.



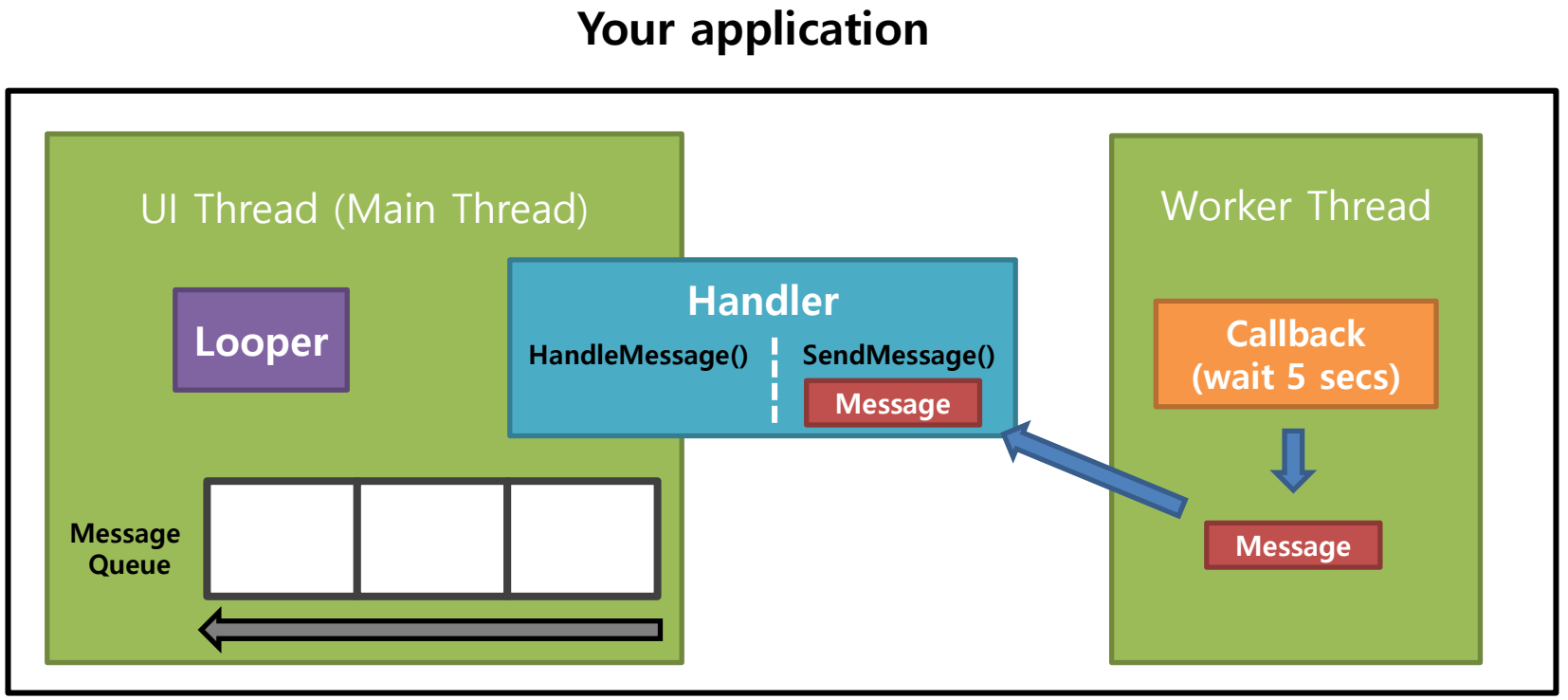
Example 2) Handler and Looper

- Using Handler you can send message or runnable instance.



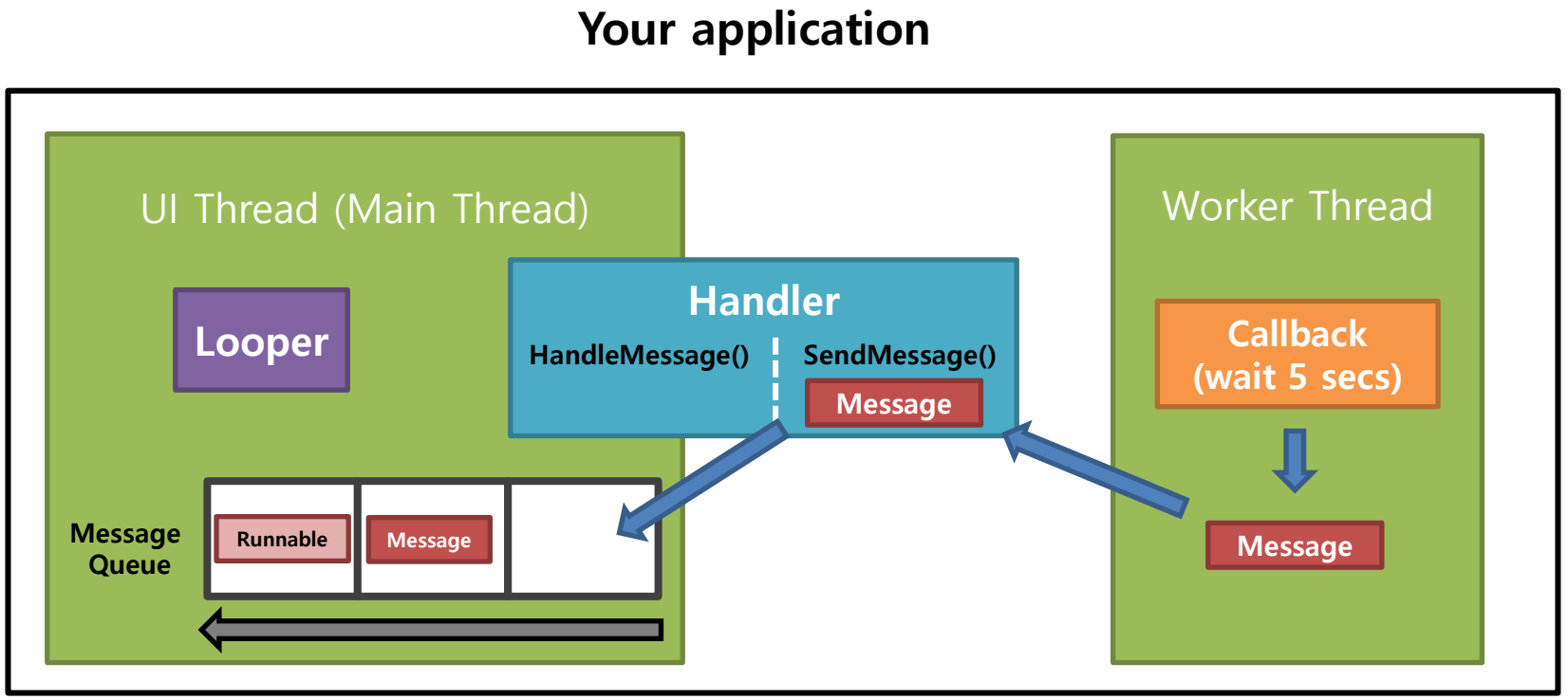
Example 2) Handler and Looper

- Using Handler you can send message or runnable instance.



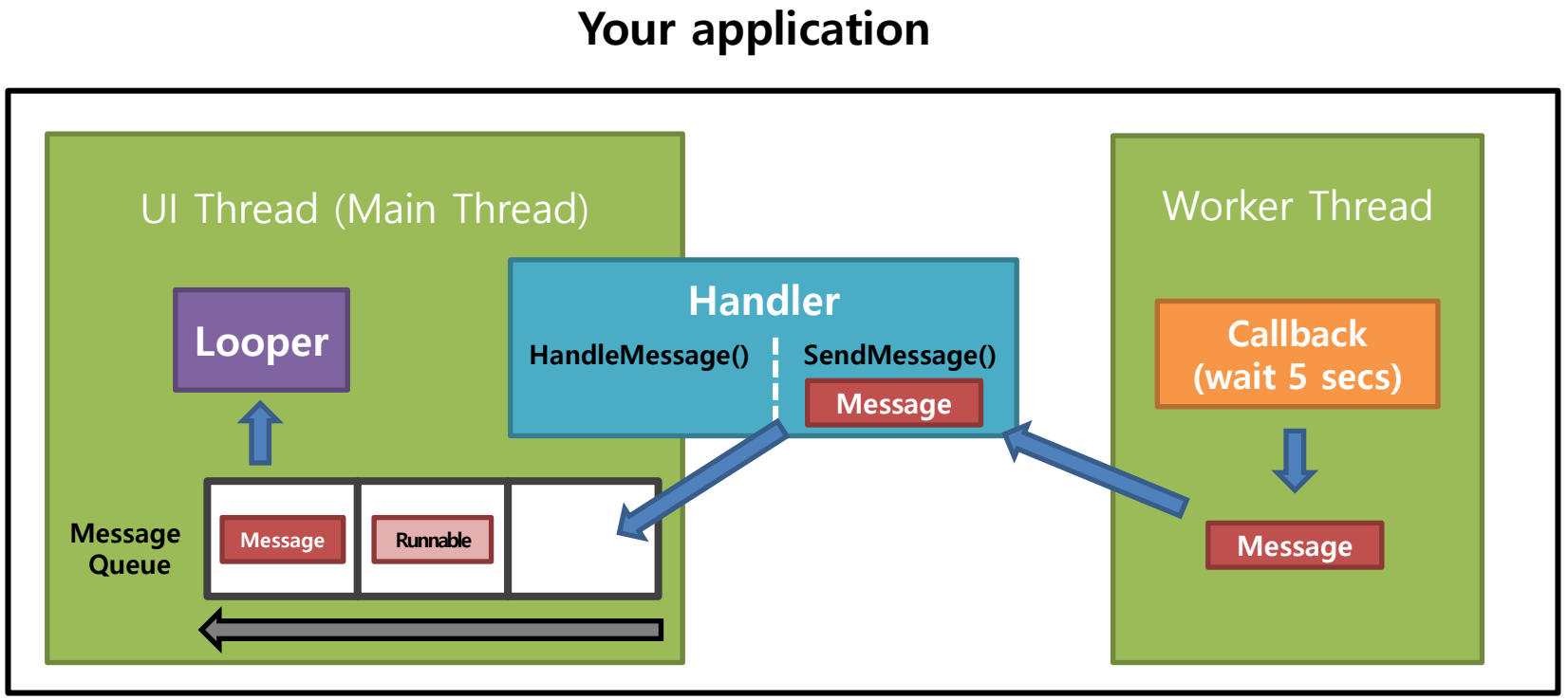
Example 2) Handler and Looper

- Using Handler you can send message or runnable instance.



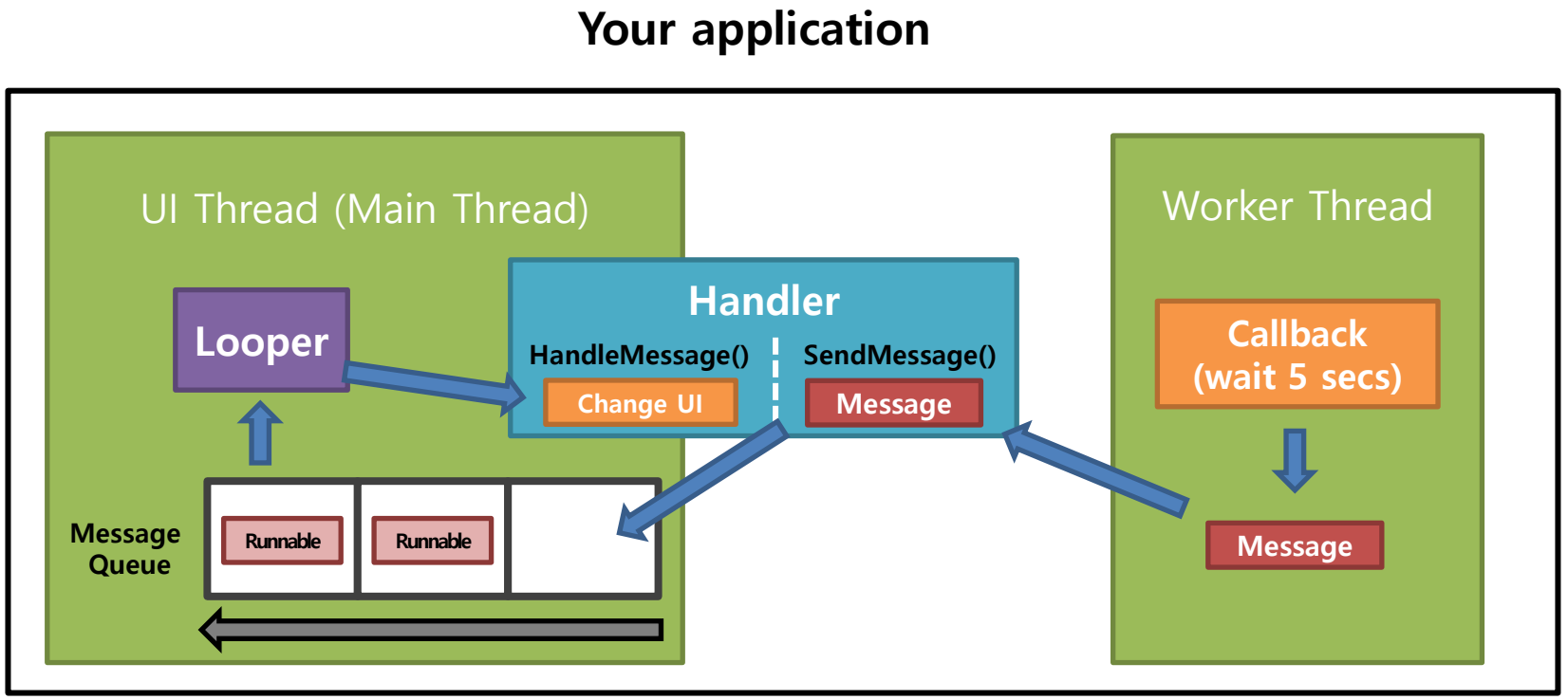
Example 2) Handler and Looper

- Using Handler you can send message or runnable instance.



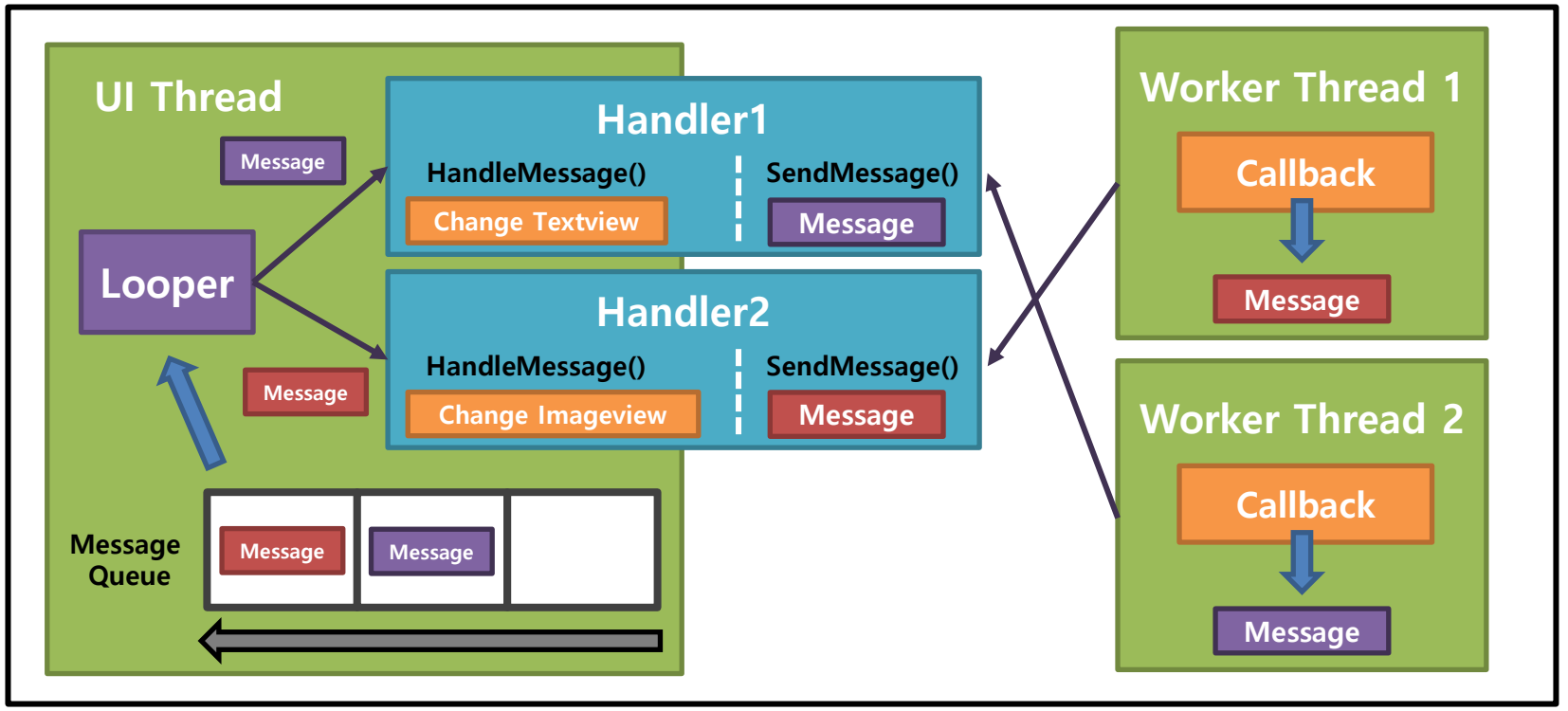
Example 2) Handler and Looper

- Using Handler you can send message or runnable instance.

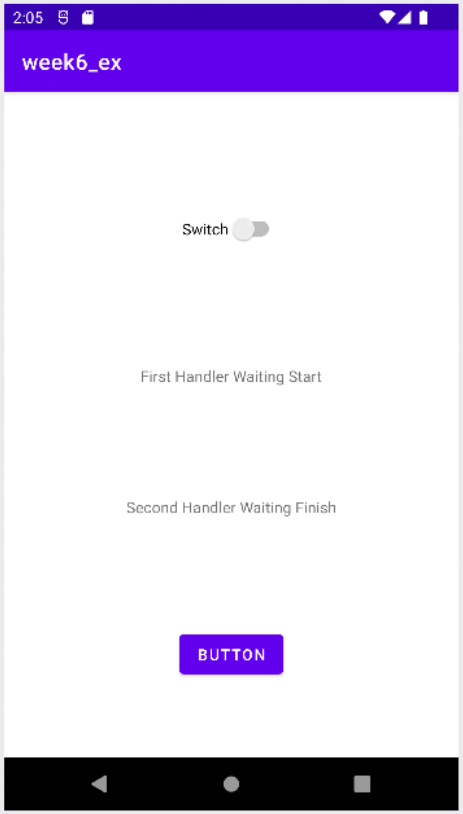


Example 2) Handler and Looper

- Of course, you can make multiple handlers!!



Example 2) Handler



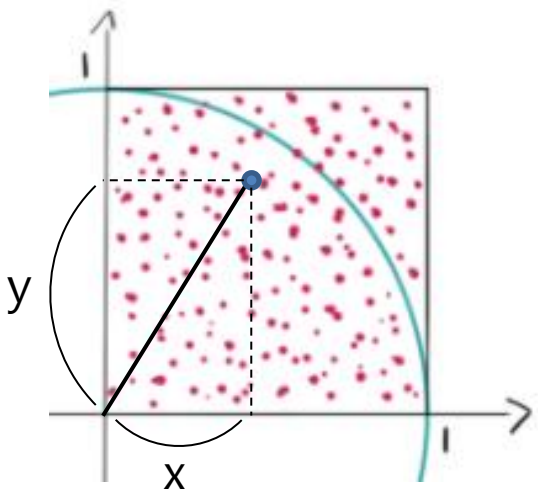

```java
public class MainActivity extends AppCompatActivity {
    TextView textView1, textView2;
    CustomHandler handler = new CustomHandler();
    CustomHandler2 handler2 = new CustomHandler2();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        CustomThread runnable = new CustomThread();
        textView1 = findViewById(R.id.textView1);
        textView2 = findViewById(R.id.textView2);
        Button btn = findViewById(R.id.button);
        btn.setOnClickListener(view -> {
            new Thread(runnable).start();
        });
    }

    …. Define Custom Thread

    …. Define Custom Handlers

}
```

```java
class CustomThread implements Runnable{
    @Override
    public void run() {
        Bundle bundle = new Bundle();
        bundle.putString("value", "Waiting Start");
        Message msg = new Message();
        msg.setData(bundle);
        handler.sendMessage(msg);

        synchronized(this) {
            try {
                wait(3000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        bundle = new Bundle();
        bundle.putString("value", "Waiting Finish");
        msg = new Message();
        msg.setData(bundle);
        handler2.sendMessage(msg);
    }
}
```

```java
class CustomHandler extends Handler{
    @Override
    public void handleMessage(@NonNull Message msg) {
        super.handleMessage(msg);
        Bundle bundle = msg.getData();
        String value = bundle.getString("value");
        textView1.setText("First Handler " + value);
    }
}

class CustomHandler2 extends Handler{
    @Override
    public void handleMessage(@NonNull Message msg) {
        super.handleMessage(msg);
        Bundle bundle = msg.getData();
        String value = bundle.getString("value");
        textView2.setText("Second Handler " + value);
    }
}
```

[Lab – Practice #6]

- Estimate Pi using Monte Carlo Simulation
  - Monte Carlo simulations are used to model the probability of different outcomes in a process that cannot easily be predicted due to the intervention of random variables.
- How to estimate pi?

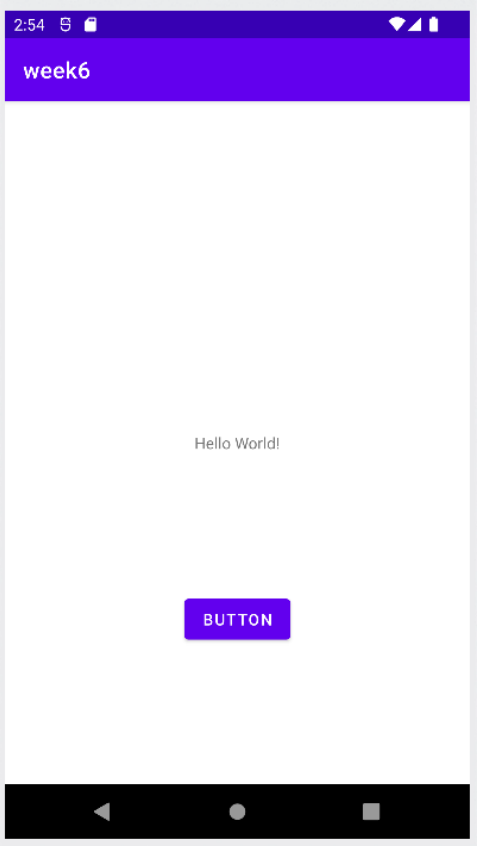


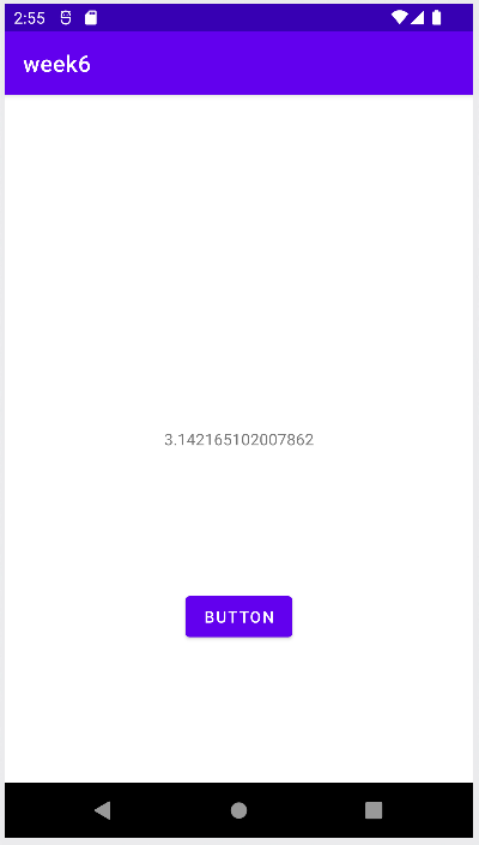
1) Get two random float values range 0~1 (x, y)
2) If (x,y) is inside of circle, $\sqrt{x^2 + y^2} \leq 1$
3) If (x,y) is outside of circle, $\sqrt{x^2 + y^2} > 1$
4) Run (1)-(3) many times
5) Calculate (number of dots inside) / (total dots)

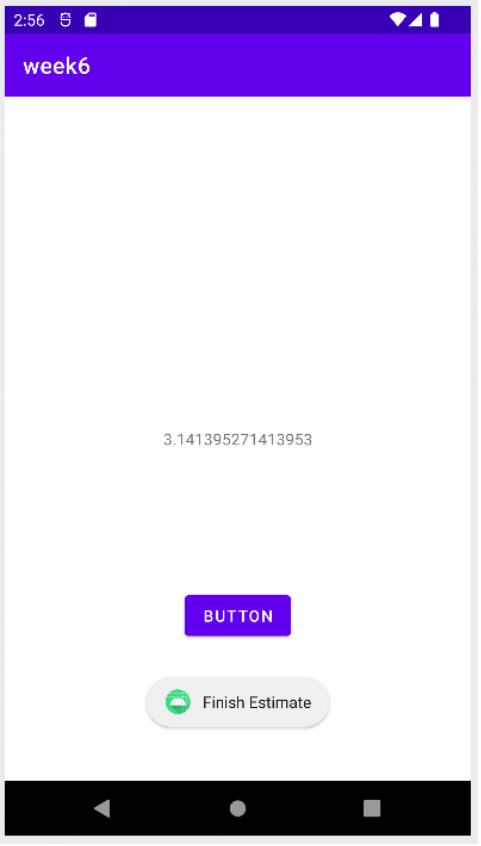
[Lab – Practice #6]

- Monte Carlo is heavy workload. Run it in thread.
- If you click button, pi estimation is started.
- Use Handler, update estimated value at every 1,000,000 samplings.
- To get random float value, use “Math.random()”
  - It will return float value between 0.0 to 1.0
- After 100,000,000 tries, finish your thread and show toast message “Finish Estimate” shortly.

[Lab – Practice #6]







[Lab – Practice #6]

- Submit your application on ICAMPUS
- File -> Export -> Export to zip…
- Change your zip to <studentID_w6>.zip