

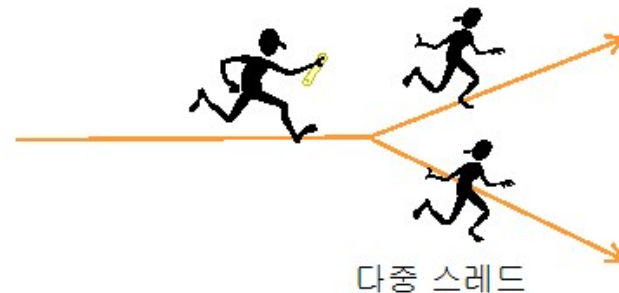
Process 와 Thread

Mobile Software

2019 Fall

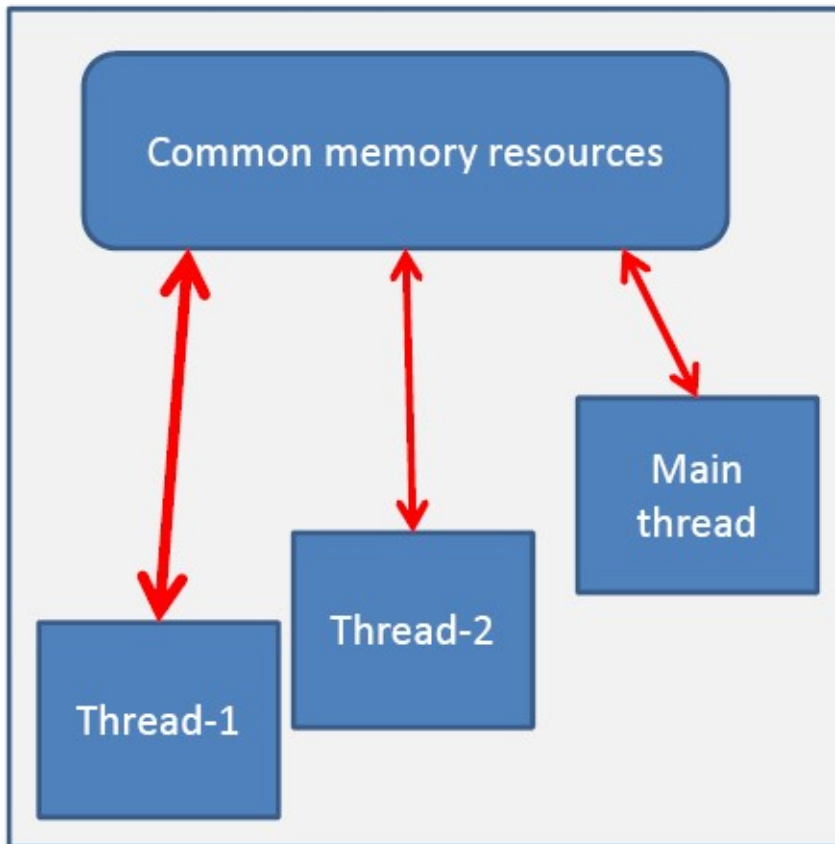
다중 스레드(Multi-threading)란?

- 하나의 애플리케이션이 동시에 여러 가지 작업을 하는 것
 - 각 작업을 thread(스레드)라고 함

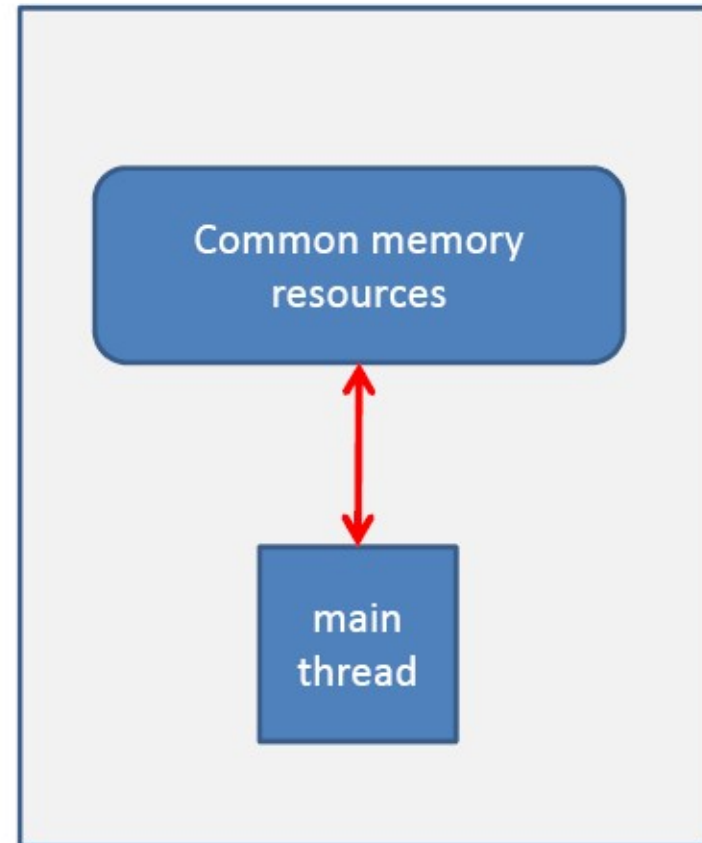


다중 스레드 (1/2)

Process 1 (Dalvik Virtual Machine 1)



Process 2 (Dalvik Virtual Machine 2)



다중 스레드 (2/2)

- 스레드는 병렬 처리(**concurrent execution**)에서 기본 요소.
- 다중 스레드는 같은 프로세스(process) 내에 함께 존재
 - H/W 자원(메모리 및 CPU)을 공유
- Application이 처리 시간이 많이 필요한 연산을 포함하고 있음에도 불구하고 user interface를 제때 처리하려면
 - 방법 1: Do expensive operations in a **background service, using notifications** to inform users about next step.
 - 방법 2: Do the slow work in a **background thread**.
 - *Main thread runs UI*, and **slow tasks** are sent to *background threads*.

UI thread와 Work thread

- **Main Thread = UI Thread**
 - User interface를 담당하는 스레드
 - Background threads are not allowed to interact with the UI.
 - **Only the main thread** can access the activity's view.
- **작업 스레드(Work thread) = Background Thread**
 - UI를 갖지 않으며,
 - background에서 동작하는 스레드

Work thread 생성 방법

- 첫 번째 방법

class *WorkThread* : *Thread* { ... }

- **Thread** 클래스를 상속받은 *WorkThread* 정의

val *thread* = **WorkThread** ()

- 기본 생성자를 사용해 Thread 객체 생성

- 두 번째 방법

class *WorkRunnable* : *Runnable* { ... }

- Runnable 인터페이스를 상속받은 *WorkRunnable* 정의
 - 이를 Thread의 생성자 인수로 전달

val *runnable* = **Thread** (*WorkRunnable*())

실습 준비

- 새 프로젝트 생성
 - Activity : **Empty Activity**
 - Application name : **Ch12_project**
 - Minimum API level : **API 26** (Oreo)
 - Activity name : **MainActivity.kt** (자동 생성)
 - Layout name
 - **activity_main.xml** (자동 생성)
- 자동 생성된 레이아웃 XML 파일은 1개
 - **activity_main.xml** 의 root layout은 **ConstraintLayout**

Thread 생성: Thread 클래스 상속

```
class MainActivity : AppCompatActivity() {
```

MainActivity.kt

```
    private val TAG = "<THREAD>"
    private var running = false
```

```
    override fun onCreate(savedInstanceState: Bundle?) { ... }
```

```
    override fun onStart() {
        super.onStart()
        val w = WorkThread()
        running = true
        w.start()
    }
```

```
    override fun onStop() {
        super.onStop()
        running = false
    }
```

```
    inner class WorkThread : Thread() { ... }
```

```
        inner class WorkThread : Thread() {
            override fun run() {
                var i = 0
                while (i < 20 && running) { ... }
            }
        }
```


실습 1: Thread 클래스 상속

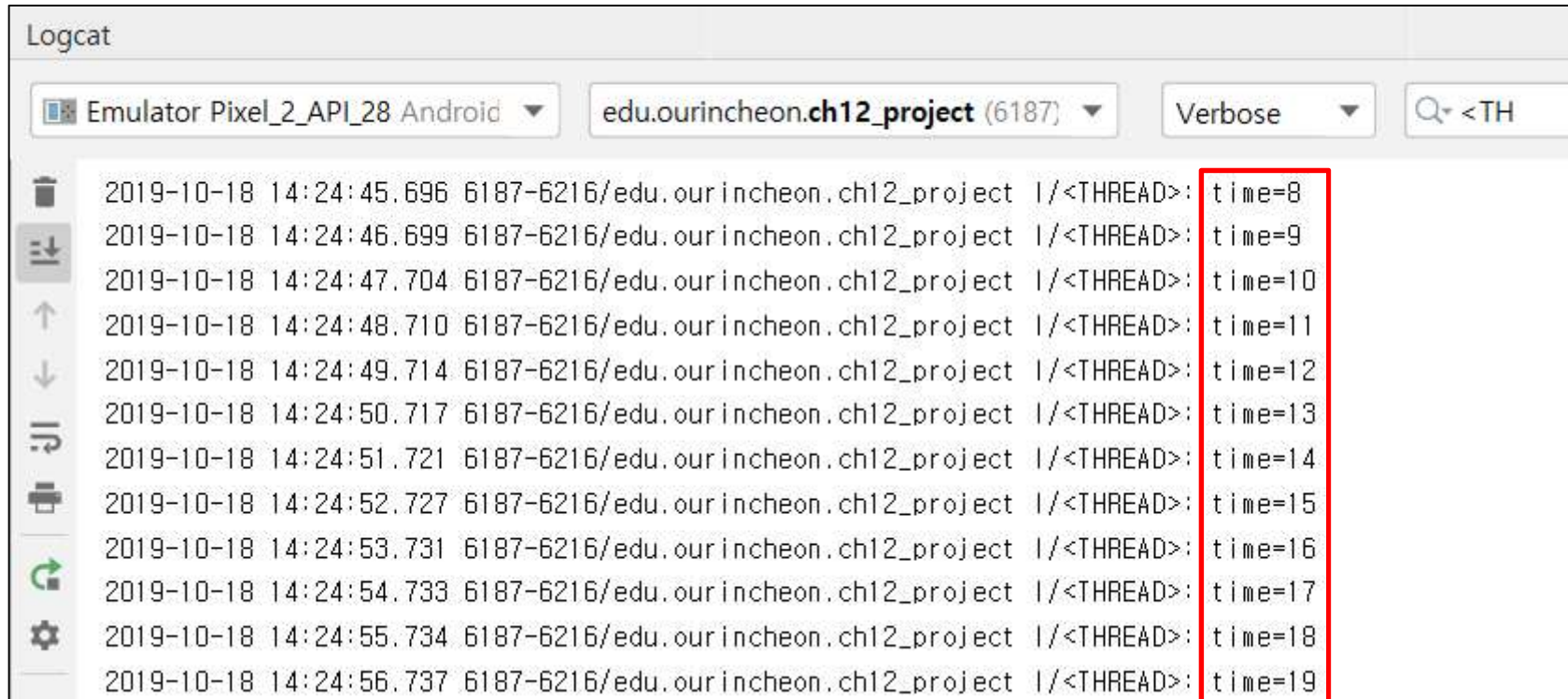
MainActivity.kt

```
inner class WorkThread : Thread() {  
    override fun run() {  
        var i = 0  
        while (i < 20 && running) {  
            try {  
                sleep(1000)  
            } catch (e: InterruptedException) {  
                Log.e(TAG, "Exception in thread: ", e)  
            }  
  
            Log.i(TAG, "time=$i")  
            i++  
        }  
    }  
}
```

자동 생성된 파일

activity_main.xml

실습 1: 실행 결과




Thread 생성: Runnable 인터페이스 구현


```
class MainActivity : AppCompatActivity() {  
  
    private val TAG = "<THREAD>"  
    private var running = false  
  
    override fun onCreate(savedInstanceState: Bundle?) {...}  
  
    inner class MyRunnable : Runnable {  
        override fun run() {...}  
    }  
  
    override fun onStart() {  
        super.onStart()  
        val w = Thread(MyRunnable())  
        running = true  
        w.start()  
    }  
  
    override fun onStop() {...}  
}
```

MainActivity.kt

실습 2: Runnable interface 구현



```
inner class MyRunnable : Runnable {  
    override fun run() {  
        var i = 0  
        while (i < 20 && running) {  
            try {  
                Thread.sleep(1000)  
            } catch (e: InterruptedException) {  
                Log.e(TAG, "Exception in thread: ", e)  
            }  
  
            Log.i(TAG, "time=$i")  
            i++  
        }  
    }  
}  
  
override fun onStart() {  
    super.onStart()  
    val w = Thread(MyRunnable())  
    running = true  
    w.start()  
}
```



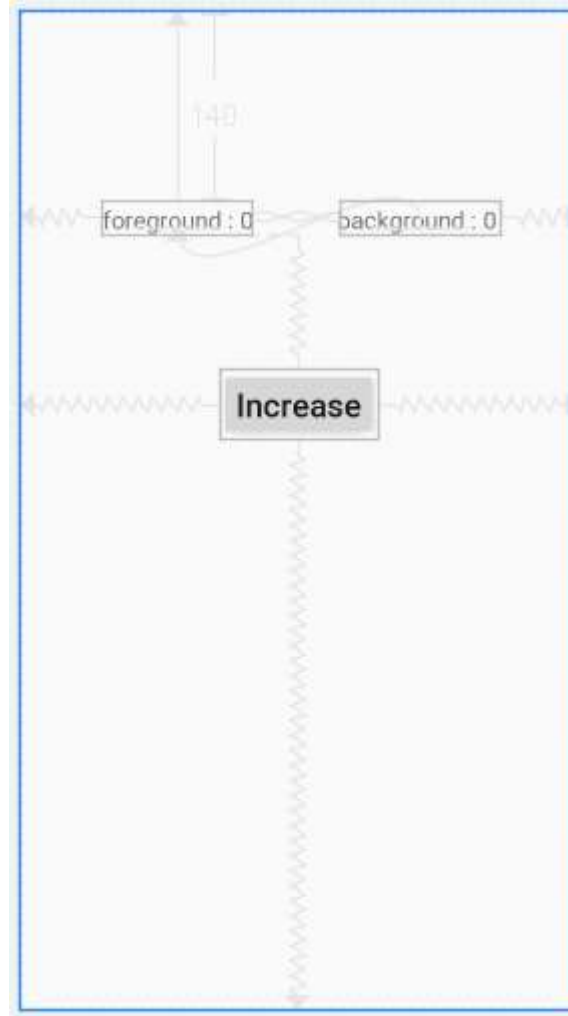
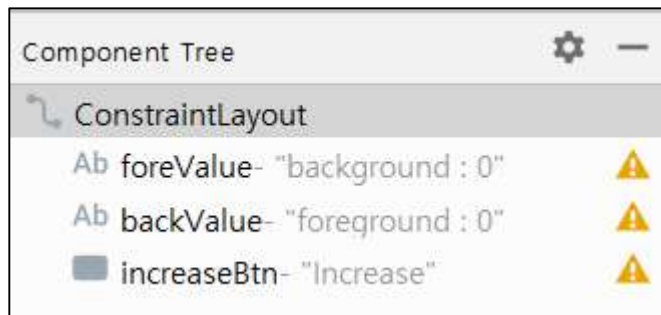
MainActivity.kt

UI thread와 작업 thread간 통신

- Work thread가 background에서 실행한 결과를 화면에 출력하고 싶으면 어떻게 할까?
 - Work thread는 UI thread에게 어떻게 정보를 전달할 수 있을까?

실습 3: Layout

activity_main.xml



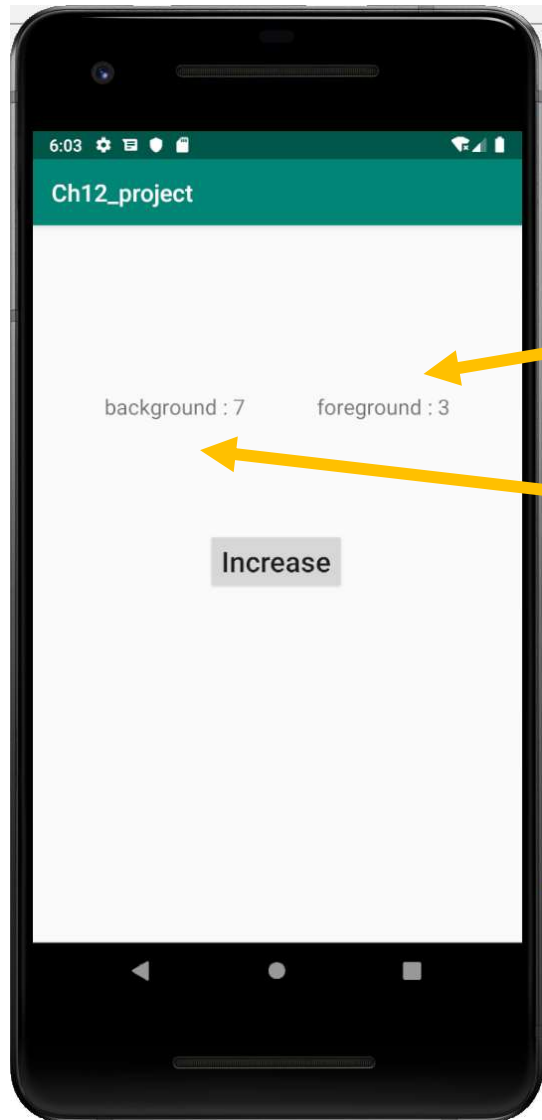
실습 3: Thread 구현

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    val t = Thread(WorkThread())  
    t.isDaemon = true  
    t.start()  
  
    increaseBtn.setOnClickListener {  
        foregroundValue++  
        foreValue.text = "foreground : $foregroundValue"  
        backValue.text = "background : $backgroundValue"  
    }  
}  
  
inner class WorkThread : Runnable {  
    override fun run() {  
        while (true) {  
            backgroundValue++  
            try {  
                Thread.sleep(1000)  
            } catch (e: InterruptedException) {  
                e.printStackTrace()  
            }  
        }  
    }  
}
```

MainActivity.kt

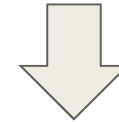
```
private var foregroundValue = 0  
private var backgroundValue = 0
```

실습 3: 실행 결과



Main thread 실행 결과
(버튼을 클릭할 때마다
값이 1씩 증가)

BackThread 실행 결과
(1초마다 값이 1씩 증가)



버튼을 누르지 않으면
BackThread 실행 결과를
알 수 없음!

실습 3: Thread에서 직접 출력하면?

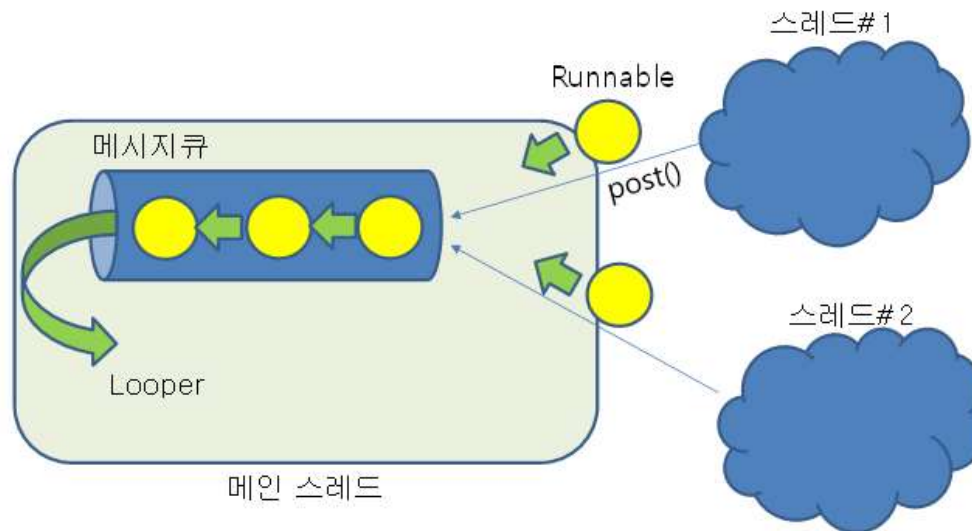
```
inner class WorkThread : Runnable {  
    override fun run() {  
        while (true) {  
            backgroundValue++  
            backValue.text = "background : $backgroundValue"  
            try {  
                Thread.sleep(1000)  
            } catch (e: InterruptedException) {  
                e.printStackTrace()  
            }  
        }  
    }  
}
```

How to solve the problem?

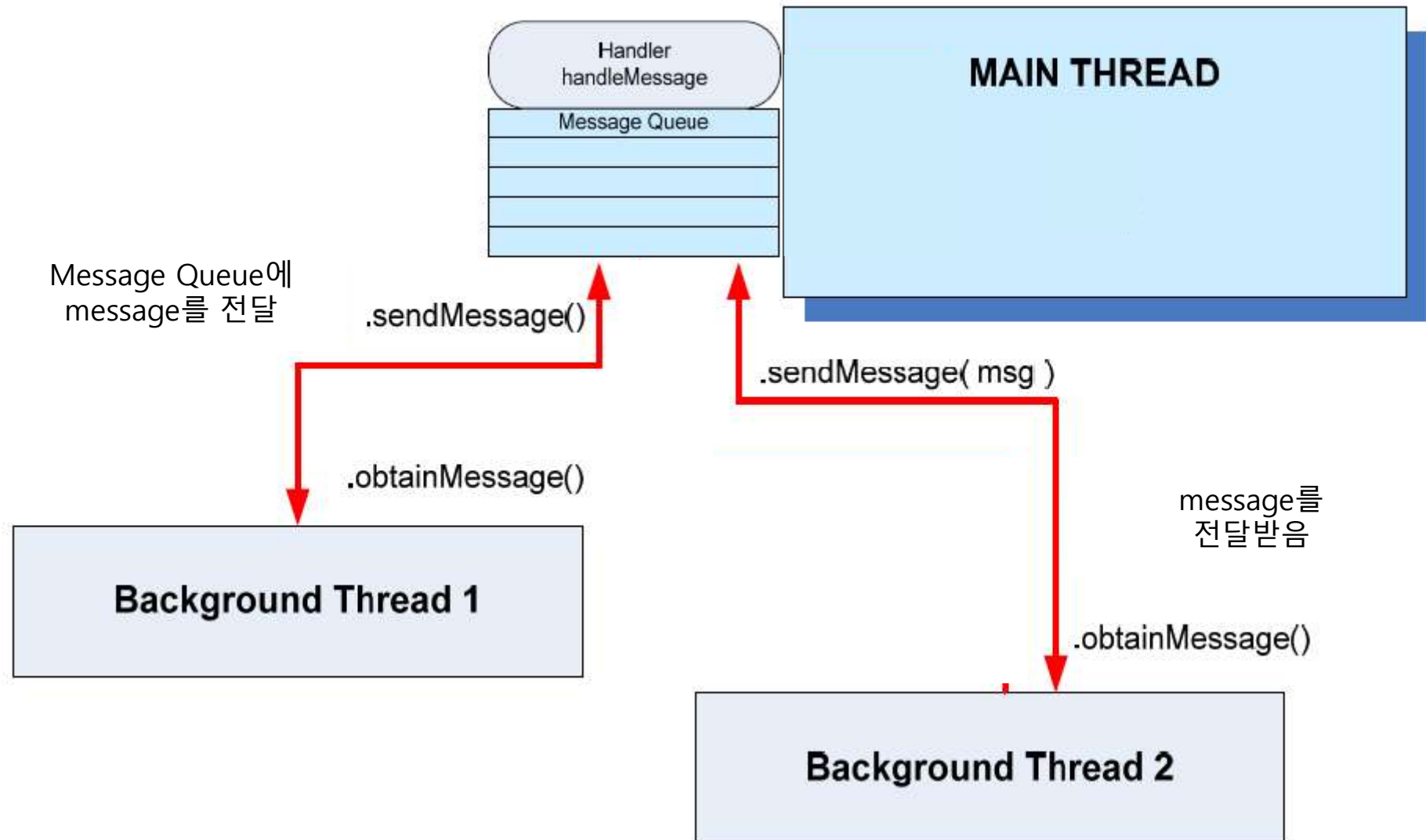
- Why it causes run-time error?
 - Background에서 동작하는 thread는 내부 연산만 처리하되,
 - main thread의 사용자 인터페이스를 바꿀 수 없다.
 - 만약 변경할 수 있게 된다면
 - 사용자가 touch해서 나타난 결과인지, Background thread의 실행 결과가 출력된 것인지 알 수 없음
- So, **why we need Handler?**
 - Thread 간에 서로 교신할 수 있는 방법은?
 - Background thread가 자신의 실행 결과를 main thread의 UI에 출력하려면?
 - Main thread는 Handler를 통해 작업 thread로부터 메시지를 전달받을 수 있다.

핸들러(Handler) 클래스 (1/3)

- A Handler allows you to send and process **Message** and **Runnable objects** associated with a **Message Queue**.
- Handler는 자신을 생성한 thread와 연결 됨
 - Handler 객체는 message queue에 저장된 message 중에서 자신이 처리해야 할 message를 알고 있다.



Handler class (2/3)



Handler class (3/3)

- 메시지 처리 : main thread가 실행
 - override fun handleMessage (msg : Message)**
 - Message 객체의 필드
 - int **what** : 메시지 내용을 설명
 - int **arg1, arg2** : 메시지 추가 정보
 - Object **obj** : 보다 복잡한 정보를 보낼 때
 - Messenger **replyTo** : 응답 받을 객체를 지정

실습 4(a) : Handler(1/2) - Runnable

MainActivity.kt

```
class MainActivity : AppCompatActivity() {  
  
    private var foregroundValue = 0  
    private var backgroundValue = 0  
    private val myHandler = MyHandler()  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        val t = Thread(WorkThread())  
        t.isDaemon = true  
        t.start()  
  
        increaseBtn.setOnClickListener {  
            foregroundValue++  
            foreValue.text = "foreground : $foregroundValue"  
            // backValue.text = "background : $backgroundValue"  
        }  
    }  
}
```

WorkThread 클래스는
Runnable 인터페이스를 상속받음

실습 4(a) : Handler(2/2) - Runnable

```
inner class MyHandler : Handler() {  
    override fun handleMessage(msg: Message?) {  
        if (msg?.what == 0) {  
            backValue.text = "background : $backgroundValue"  
        }  
    }  
}  
  
inner class WorkThread : Runnable {  
    override fun run() {  
        while (true) {  
            backgroundValue++  
            myHandler.sendMessage(0)  
            try {  
                Thread.sleep(1000)  
            } catch (e: InterruptedException) {  
                e.printStackTrace()  
            }  
        }  
    }  
}
```

실습 4(b) : Handler 구현 - Thread



```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    val t = WorkThread()  
    t.isDaemon = true  
    t.start()  
  
    increaseBtn.setOnClickListener {...}  
}  
  
inner class MyHandler : Handler() {...}  
  
inner class WorkThread : Thread() {  
    override fun run() {  
        while (true) {  
            backgroundValue++  
            myHandler.sendMessage(0)  
            try {  
                sleep(1000)  
            } catch (e: InterruptedException) {  
                e.printStackTrace()  
            }  
        }  
    }  
}
```

WorkThread 클래스는
Thread 클래스를 상속받음



실습 4(c) : Handler 구현 – sendMessage

MainActivity.kt

```
inner class MyHandler : Handler() {  
    override fun handleMessage(msg: Message?) {  
        if (msg == null) return  
        var bundle = msg.data  
        backValue.text = "background : ${bundle.getInt("COUNT")}"  
    }  
}  
  
inner class WorkThread : Thread() {  
    override fun run() {  
        while (true) {  
            backgroundValue++  
  
            val msg = myHandler.obtainMessage()  
            val bundle = Bundle()  
            bundle.putInt("COUNT", backgroundValue)  
            msg.data = bundle  
            myHandler.sendMessage(msg)  
            try {  
                sleep(1000)  
            } catch (e: InterruptedException) {  
                e.printStackTrace()  
            }  
        }  
    }  
}
```

실습 5: runnable 객체 전달

```
private val myHandler = Handler()

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

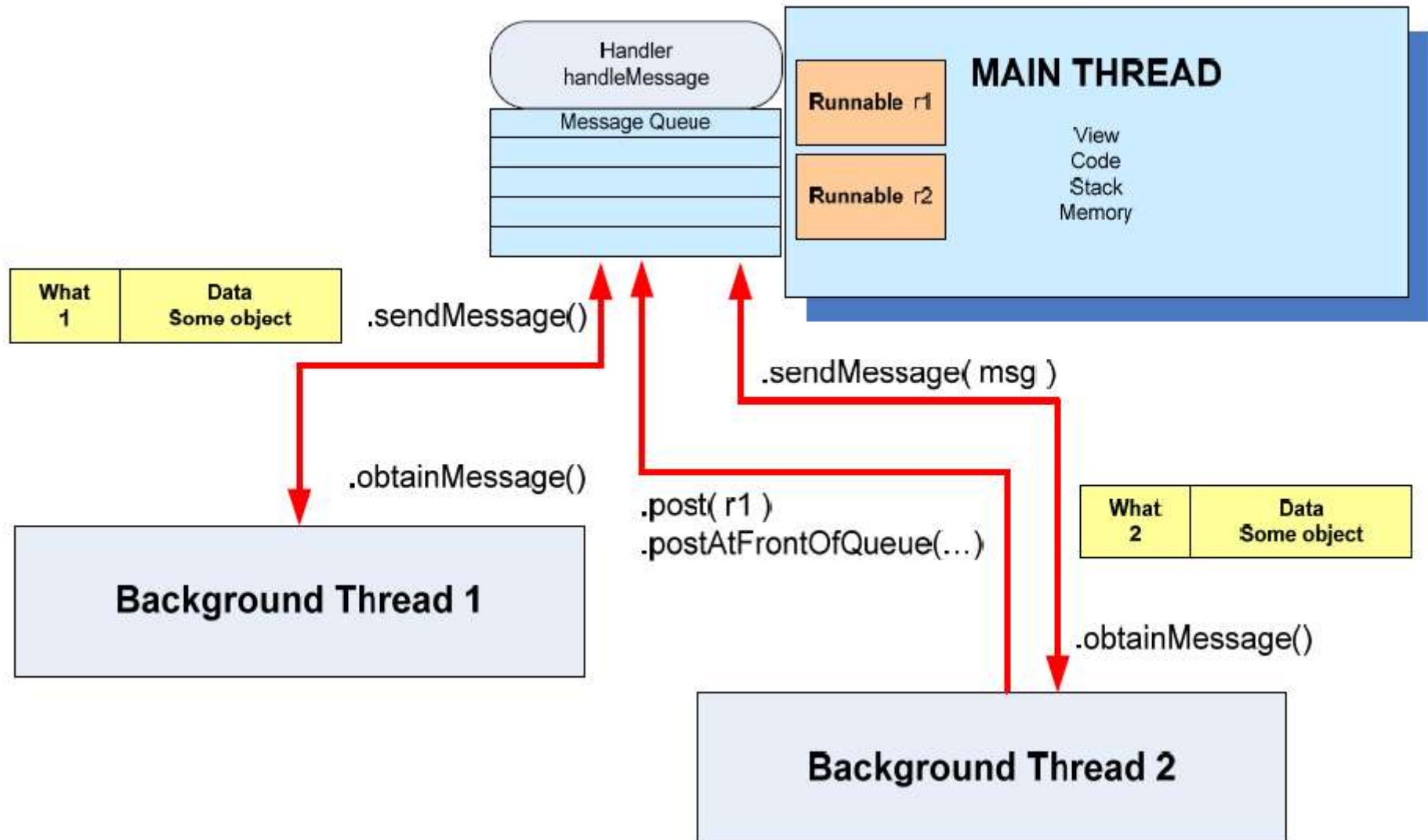
    val t = Thread(WorkThread())
    t.isDaemon = true
    t.start()

    inner class MyTask : Runnable {
        override fun run() {
            backValue.text = "background : $backgroundValue"
        }
    }
}
```

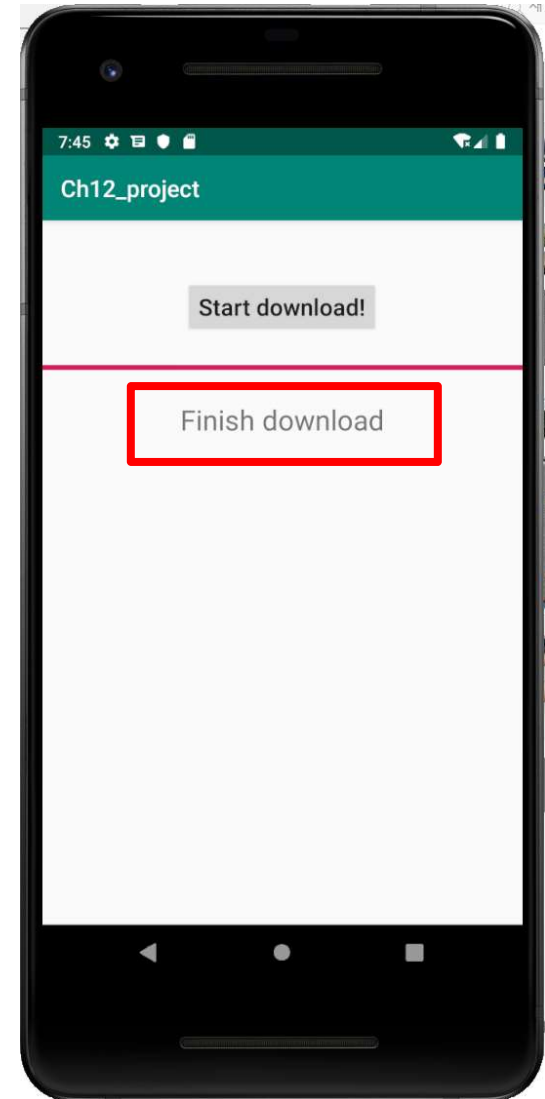
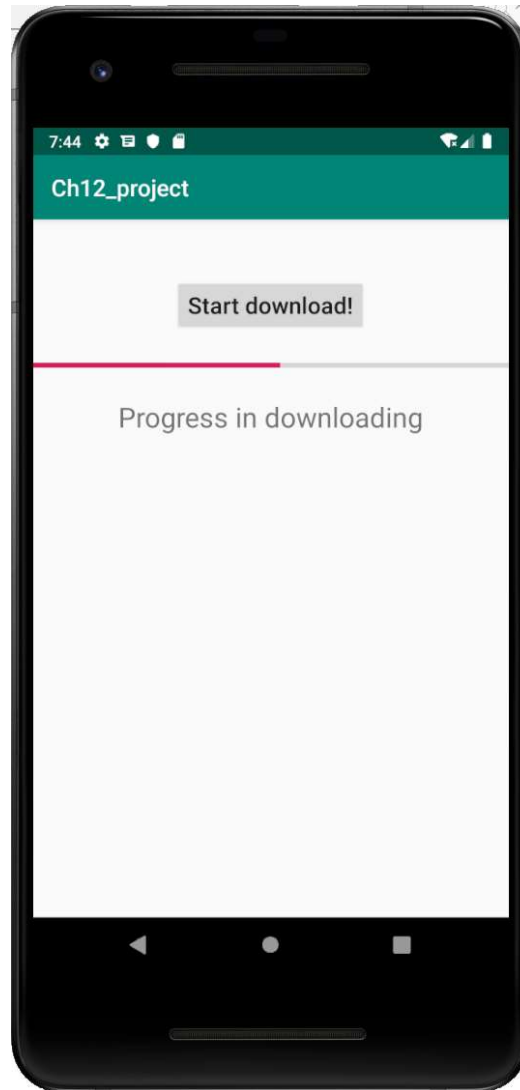
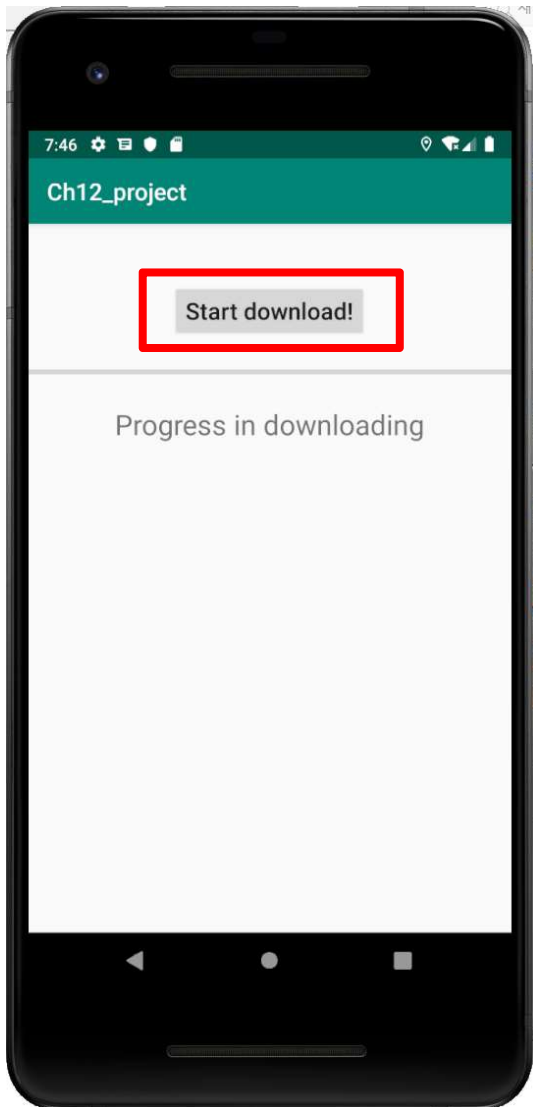
```
inner class WorkThread : Runnable {
    override fun run() {
        while (true) {
            backgroundValue++
            myHandler.post(MyTask())
            try {
                Thread.sleep(1000)
            } catch (e: InterruptedException) {
                e.printStackTrace()
            }
        }
    }
}
```

MyTask 클래스는
Runnable 인터페이스를 상속받음

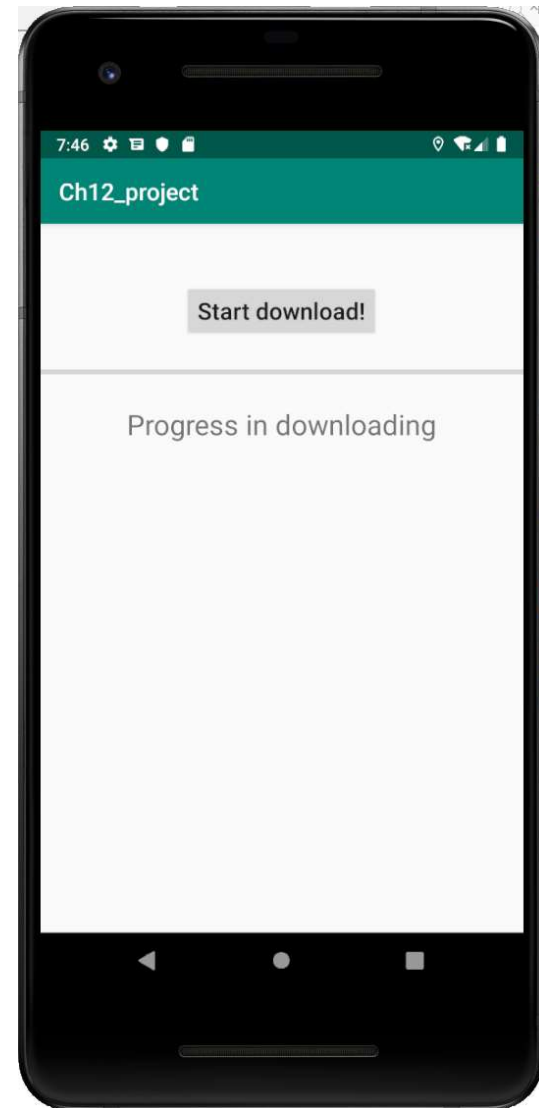
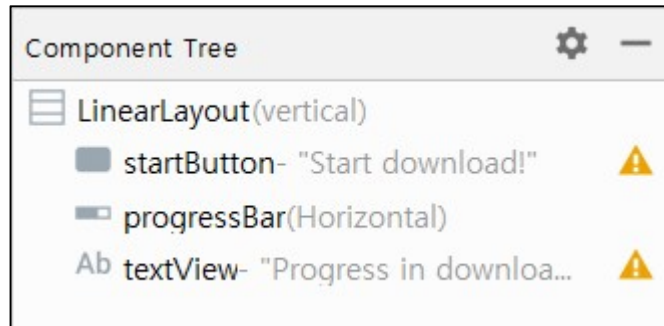
Handler 와 Message Queue



실습 6: Runnable 객체 전달



실습 6: ProgressBar – XML Layout



실습 6: ProgressBar(1/2)

```
class MainActivity : AppCompatActivity() {  
  
    private val myHandler = Handler()  
    var IncrementBy = 1  
    var max = 0  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        max = progressBar.max  
  
        startButton.setOnClickListener {  
            val t = Thread(WorkThread())  
            t.isDaemon = true  
            t.start()  
        }  
    }  
}
```

MainActivity.kt

실습 6: ProgressBar(2/2)

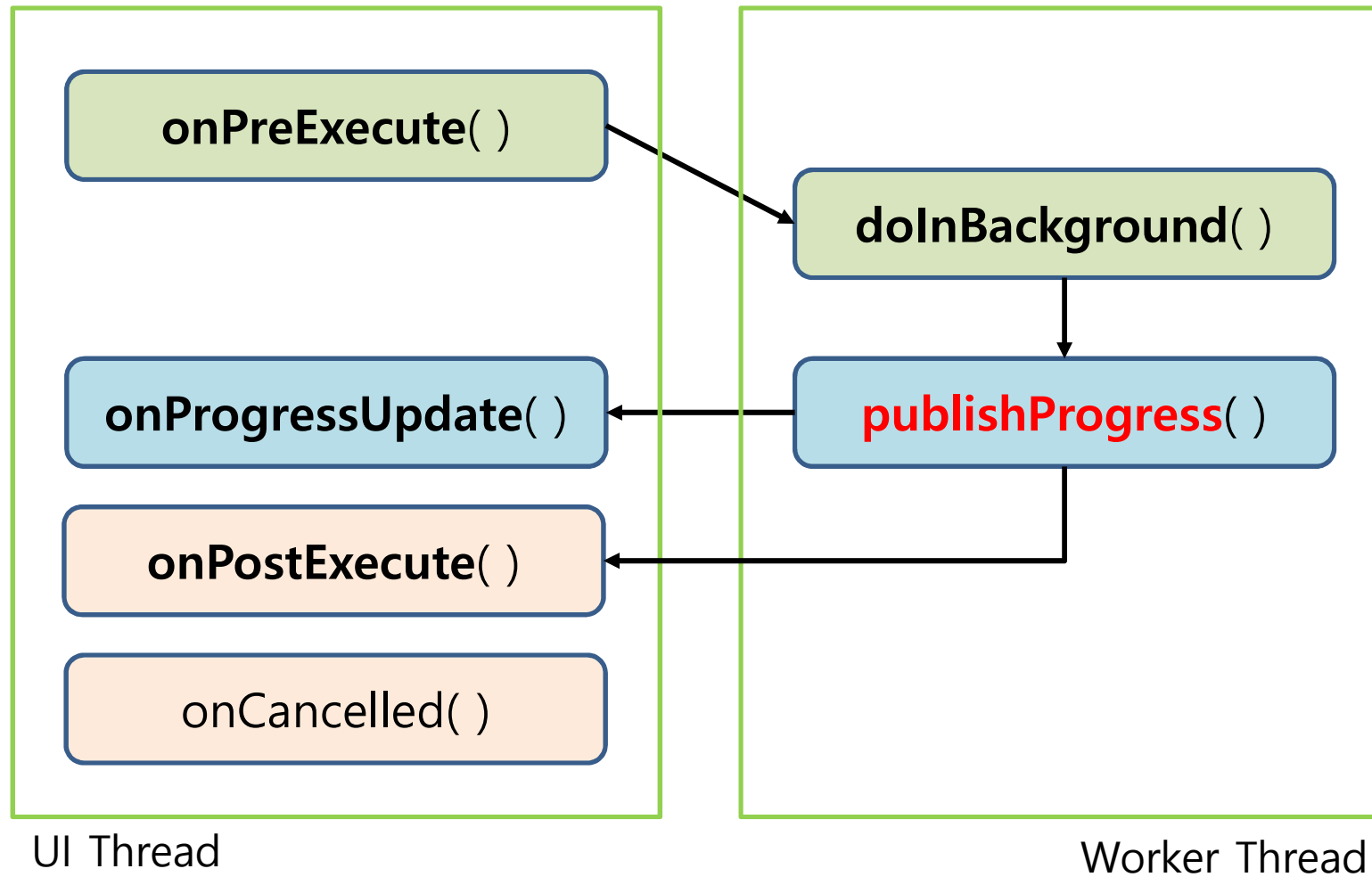
```
inner class MyTask : Runnable {  
    override fun run() {  
        if (progressBar.progress == max) {  
            textView.text = "Finish download"  
        } else {  
            progressBar.incrementProgressBy(IncrementBy)  
        }  
    }  
}  
  
inner class WorkThread : Runnable {  
    override fun run() {  
        while (progressBar.progress < max) {  
            try {  
                Thread.sleep(100)  
            } catch (e: InterruptedException) {  
                e.printStackTrace()  
            }  
            myHandler.post(MyTask())  
        } // end of while  
    }  
}
```



AsyncTask 클래스(1/3)

- 1.5 버전부터 추가된 클래스
 - 작업 thread와 관련된 복잡한 부분을 쉽게 처리해주는 클래스
 - Background 수행 결과를 Handler를 거치지 않고서도 UI Thread에 전달
- 짧은 시간 내에 처리 가능한 작업에 최적화
 - 처리 시간이 긴 작업을 수행해야 한다면 java.util.concurrent 패키지의 Executor나 ThreadPoolExecutor, FutureTask 등을 사용
- 3개의 Generic Type parameter가 필요
 - onPreExecute, doInBackground, onProgressUpdate, onPostExecute 등 4 단계로 작업을 처리

AsyncTask 클래스(2/3)

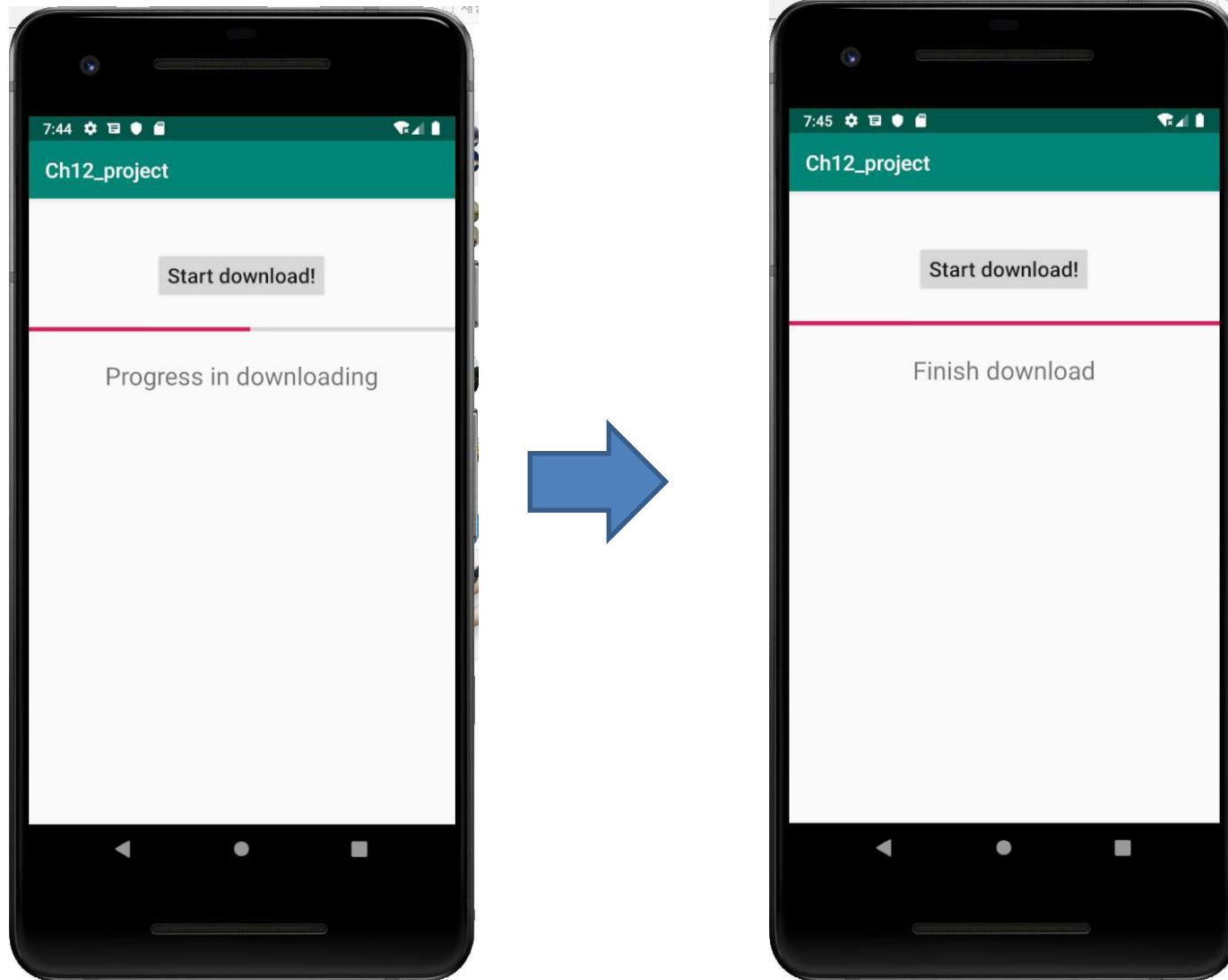


AsyncTask 클래스(3/3)

```
class MyTask : AsyncTask<Int, Int, Int>()
```

- 3개의 generic parameter
 - **Params** : 실행 중 전달되는 값
 - **Progress** : 작업 진행 정도를 나타내는 값
 - **Result** : 작업을 마친 후 결과 값
- Method
 - **doInBackground** : 작업 thread에서 실행
 - return 값은 onPostExecute로 전달됨.
 - 언제든지 **publishProgress** 를 호출하여 UI thread에서 onProgressUpdate 를 실행할 수 있음
- 가변 인수(varargs)
 - **doInBackground** (vararg params: Int?) : Int
 - 첫 번째 매개변수만 표시하고 나머지는 생략할 때 사용

실습 7: AsyncTask



실습 7: AsyncTask – Activity (1/3)

```
class MainActivity : AppCompatActivity() {  
  
    private var incrementBy = 1  
    private var max = 0  
  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
  
        startButton.setOnClickListener {  
            val task = MyTask().execute()  
        }  
    }  
}
```

실습 7: AsyncTask – Activity (2/3)

```
private inner class MyTask : AsyncTask<Int, Int, Int>() {  
  
    var curProgress = 0  
  
    override fun doInBackground(vararg params: Int?): Int {  
        while (curProgress < max) {  
            try {  
                Thread.sleep(100)  
            } catch (e: InterruptedException) {  
                e.printStackTrace()  
            }  
            curProgress += incrementBy  
            publishProgress(curProgress)  
        }  
        return curProgress  
    }  
  
    override fun onProgressUpdate(vararg values: Int?) {...}  
  
    override fun onPostExecute(result: Int?) {...}  
  
    override fun onPreExecute() {...}  
}
```

The diagram illustrates the flow of data from the `publishProgress` call in the `doInBackground` method to the `onProgressUpdate` method. A red box highlights the `publishProgress(curProgress)` call. Three red arrows originate from this box: one points to the `publishProgress` method name, another points to the `curProgress` argument, and a third points to the `onProgressUpdate` method signature, indicating that the progress value is passed to the UI thread via this callback.

실습 7: AsyncTask – Activity (3/3)

```
private inner class MyTask : AsyncTask<Int, Int, Int>() {  
  
    var curProgress = 0  
  
    override fun doInBackground(vararg params: Int?): Int {  
        ...  
    }  
  
    override fun onProgressUpdate(vararg values: Int?) {  
        super.onProgressUpdate(*values)  
        progressBar.progress = values[0] ?: 0  
        textView.text = "Current value : ${progressBar.progress}"  
    }  
  
    override fun onPostExecute(result: Int?) {  
        super.onPostExecute(result)  
        textView.text = "Finish download!"  
    }  
  
    override fun onPreExecute() {  
        super.onPreExecute()  
        max = progressBar.max  
    }  
}
```