



안드로이드 앱 프로그래밍

Chapter 10

스레드와 핸들러 이해하기



이번 장에서는 무엇을 다룰까요?



동시에 여러 작업을 할 수 있는 방법을 알고 싶어요.



- 스레드를 사용하는 방법에 대해 알아보을까요?
- 핸들러라는 것을 이용해 UI를 업데이트해 볼까요?
- AsyncTask 객체를 만들어 쉽게 실행하는 방법을 알아보을까요?
- 스레드를 이용해 애니메이션을 만들어 볼까요?





이번 장에서는 무엇을 다룰까요?



스레드 결과를 화면에 보여주고 싶어요.

- 핸들러 이해하기



스레드를 다루는 방법을 구체적으로 알고 싶어요.

- 일정시간 후에 실행하기
- 스레드로 메시지 전송하기



하나의 클래스에서 스레드 실행과 화면 표시를 같이 할 수 있나요?

- AsyncTask 사용하기



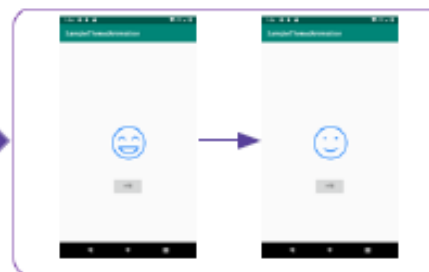
스레드를 이용해서 애니메이션을 만들 수 있나요?

- 스레드로 애니메이션 만들기

스레드에서 화면에 보여주기



스레드로 애니메이션 만들기





강의 주제

스레드와 핸들러에 대해 이해하기



1

핸들러 이해하기

2

일정시간 후에 실행하기

3

스레드로 메시지 전송하기

4

AsyncTask 사용하기

5

스레드로 애니메이션 만들기

1.

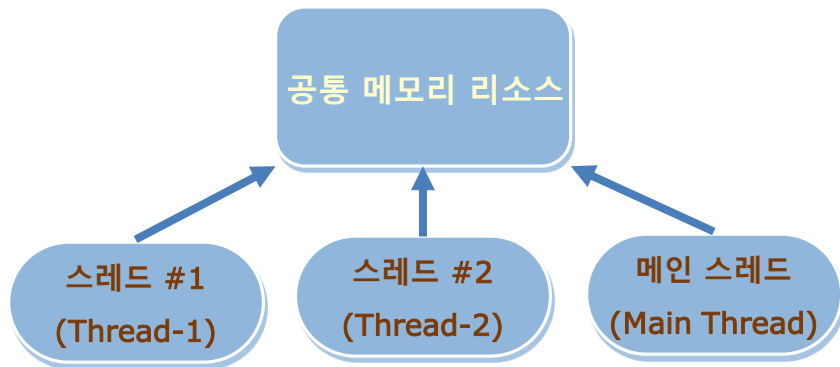
핸들러 이해하기



멀티 스레드



(1) 프로젝트 생성 시



(2) 별도의 스레드 생성 시

[멀티스레드 시스템에서 시스템에서
공통 메모리 리소스 접근]

• 메인 액티비티

- 애플리케이션이 실행될 때 하나의 프로세스에서 처리
- 이벤트를 처리하거나 필요한 메소드를 정의하여 기능을 구현하는 경우에도 동일한 프로세스 내에서 실행

• 문제점

- 대기 시간이 길어지는 네트워크 요청 등의 기능을 수행할 때는 화면에 보이는 UI도 멈춤 상태로 있게 됨

• 해결 방안

- 하나의 프로세스 안에서 여러 개의 작업이 동시 수행되는 멀티 스레드 방식을 사용

• 멀티 스레드

- 같은 프로세스 안에 들어 있으면서 메모리 리소스를 공유하게 되므로 효율적인 처리가 가능
- 동시에 리소스를 접근할 경우 데드락(DeadLock) 발생



핸들러 사용하기

시나리오	설 명
서비스 사용	백그라운드 작업은 서비스로 실행하고 사용자에게는 알림 서비스를 이용해 알려줌 만약 메인 액티비티로 결과값을 전달하고 이를 이용해 다른 작업을 수행하고자 한다면 브로드캐스팅을 이용하거나 하여 결과값을 전달할 수 있음
스레드 사용	스레드는 동일 프로세스 내에 있기 때문에 작업 수행의 결과를 바로 처리할 수 있음 그러나 UI 객체는 직접 접근할 수 없으므로 핸들러(Handler) 객체를 사용함



표준 자바에서 스레드 사용 방법

- 스레드는 new 연산자를 이용하여 객체를 생성한 후 start() 메소드를 호출하면 시작함
- Thread 클래스에 정의된 생성자는 크게 파라미터가 없는 경우와 Runnable 객체를 파라미터로 가지는 두 가지로 구분함

```
running = true;
```

```
Thread thread1 = new  
BackgroundThread();  
thread1.start();
```




표준 자바에서 스레드 사용 방법 (계속)

- 버튼을 누르면 value 변수에 들어있는 정수 타입의 값을 텍스트뷰에 보여줌
- 스레드를 'BackgroundThread'라는 이름으로 정의

```
class BackgroundThread extends Thread {  
    public void run() {  
        while (running) {  
            try {  
                Thread.sleep(1000);  
                value++;  
            } catch (InterruptedException ex) {  
                Log.e("SampleJavaThread", "Exception in thread.", ex);  
            }  
        }  
    }  
}
```



메시지 전송하여 실행하기

• 메인 스레드

- 애플리케이션 객체인 액티비티, 브로드캐스트 수신자 등과 새로 만들어지는 윈도우를 관리하기 위한 메시지 큐(Message Queue)를 실행함

• 메시지 큐 : Message Queue

- 순차적으로 코드를 수행함

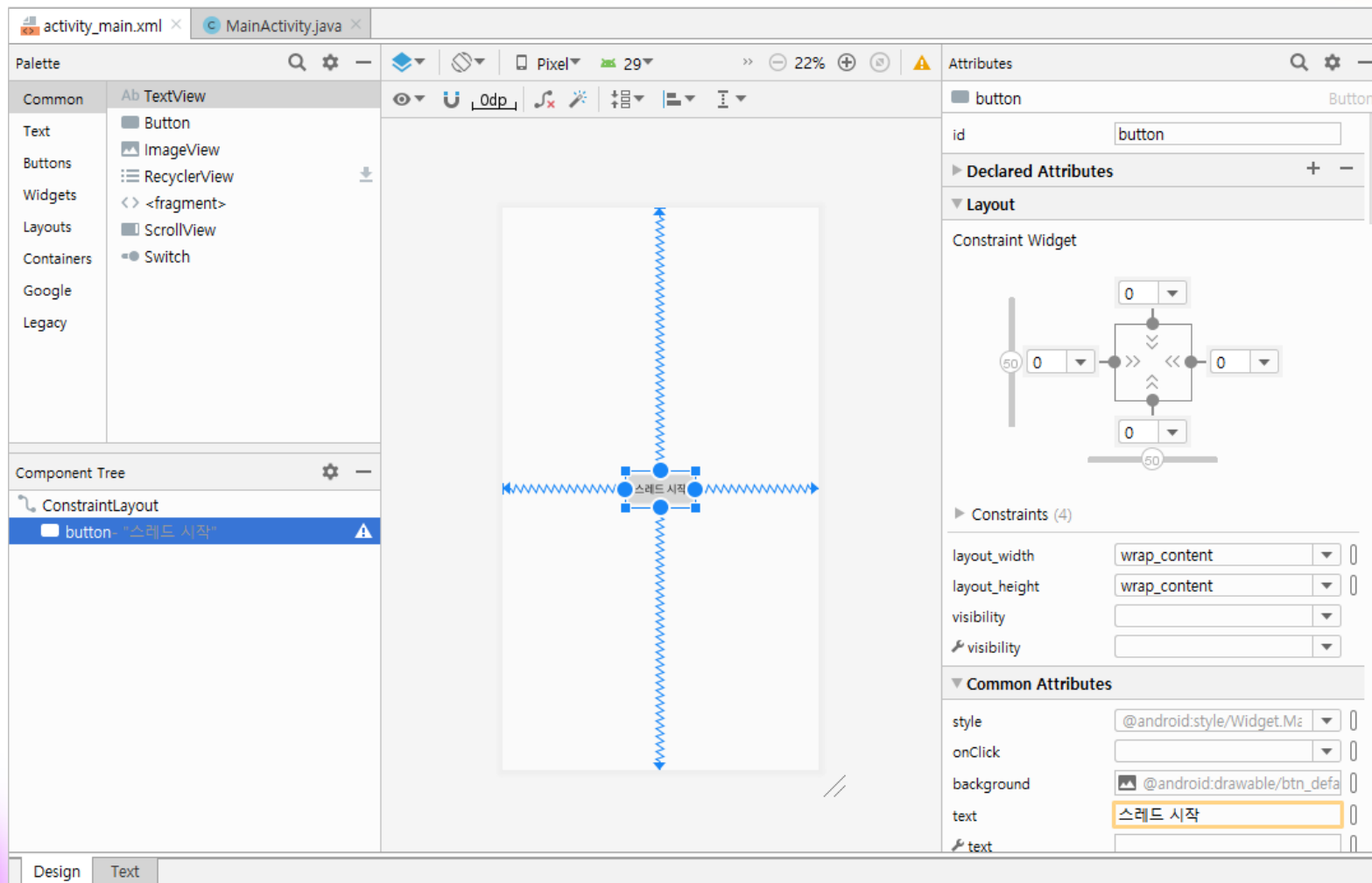
• 핸들러 : Handler

- 메시지 큐를 이용해 메인 스레드에서 처리할 메시지를 전달하는 역할을 담당함
- 특정 메시지가 미래의 어떤 시점에 실행되도록 스케줄링 할 수 있음



스레드 사용하기

- SampleThread 프로젝트 만들고 화면 레이아웃 구성





스레드 사용하기

- 메인 액티비티에서 버튼 클릭 시 스레드 동작하도록 코드 추가

참조파일 SampleThread>/app/java/org.techtown.thread/MainActivity.java

```
public class MainActivity extends AppCompatActivity {  
    int value = 0;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        Button button = findViewById(R.id.button);  
        button.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                BackgroundThread thread = new BackgroundThread();  
                thread.start();  
            }  
        });  
    }  
}
```

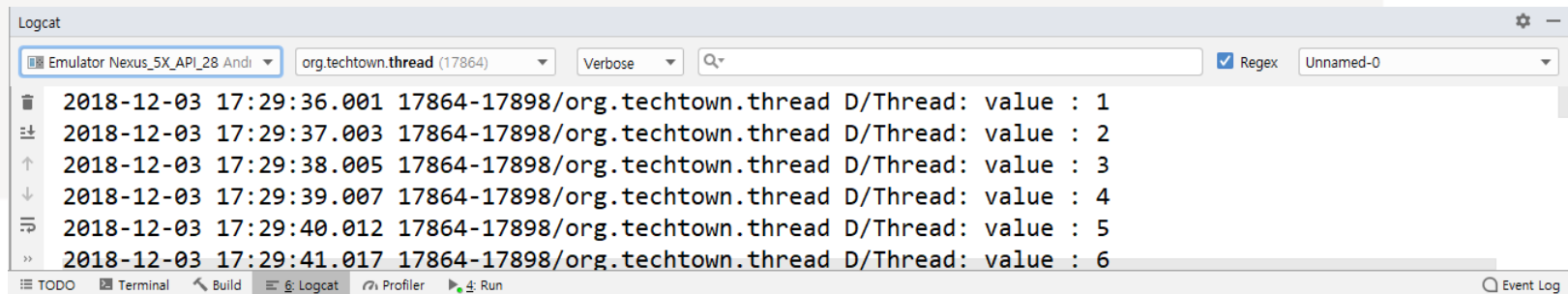
스레드 객체 생성하고 시작시키기



스레드 사용하기

- 메인 액티비티에서 버튼 클릭 시 스레드 동작하도록 코드 추가
- 앱 실행하고 버튼 누르면 로그 정상 출력됨

```
class BackgroundThread extends Thread {  
    public void run() {  
        for (int i = 0; i < 100; i++) {  
            try {  
                Thread.sleep(1000);  
            } catch (Exception e) {}  
  
            value += 1;  
            Log.d("Thread", "value : " + value);  
        }  
    }  
}
```





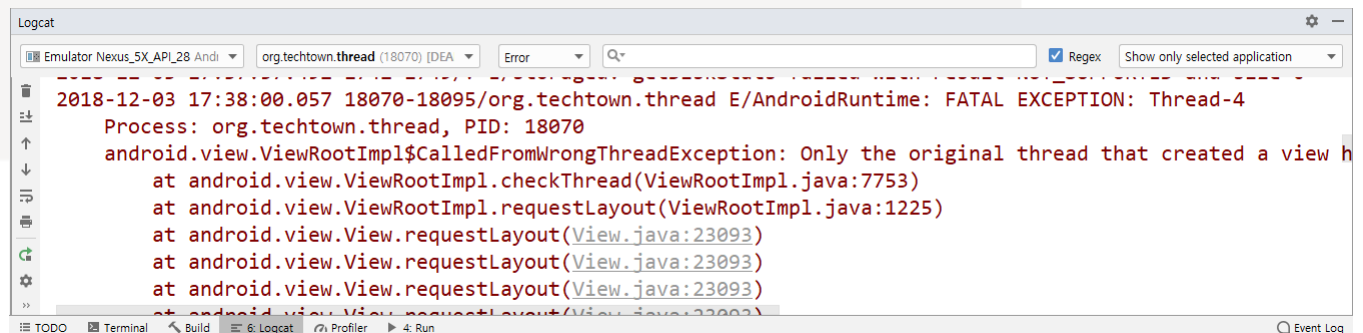
스레드 사용하기

- 스레드 안에서 텍스트뷰에 결과를 출력하도록 수정
- 앱 실행하면 오류 발생

```
class BackgroundThread extends Thread {  
    public void run() {  
        for (int i = 0; i < 100; i++) {  
            try {  
                Thread.sleep(1000);  
            } catch (Exception e) {}  
  
            value += 1;  
            Log.d("Thread", "value : " + value);  
        }  
    }  
}
```

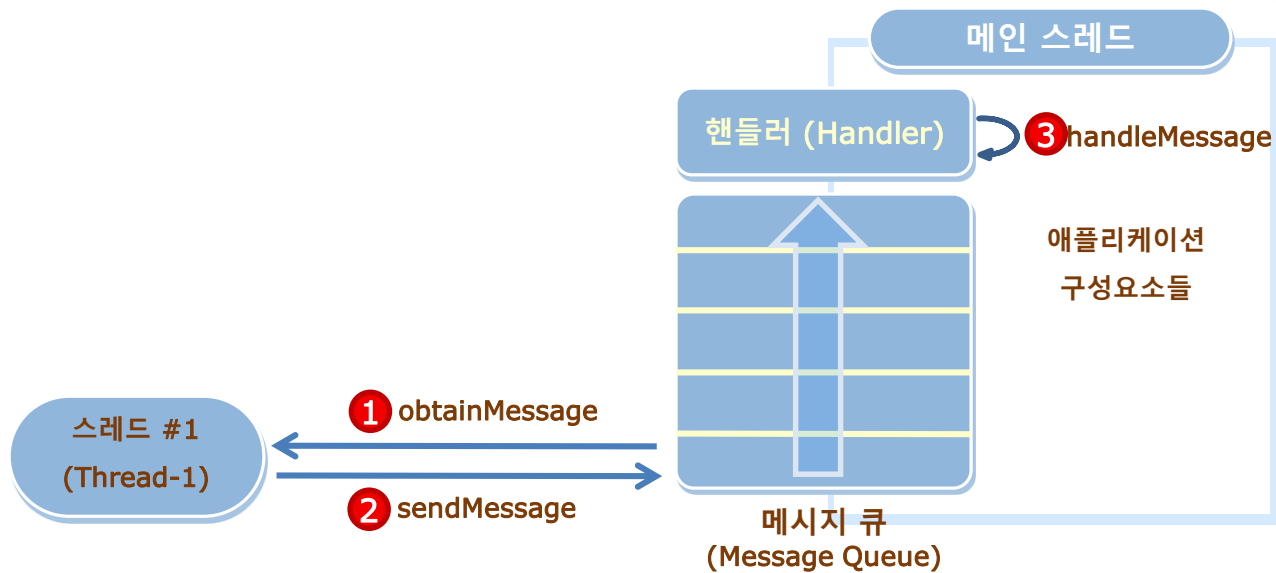
```
    textView.setText("value 값: " + value);  
}  
}
```

→ 스레드 안에서 텍스트뷰의 setText() 메서드 호출하기





핸들러로 메시지 전송하여 실행하는 구조



[핸들러를 사용할 때 필요한 세 가지 단계]

- **obtainMessage()**

- 호출의 결과로 메시지 객체를 리턴받게함

- **sendMessage()**

- 메세지큐에 넣음

- **handleMessage()**

- 메소드에 정의된 기능이 수행됨
- 코드가 수행되는 위치는 새로 만든 스레드가 아닌 메인 스레드가 됨



핸들러 사용하기

- Handler 클래스 상속하여 새로운 클래스 정의
- 전달된 메시지 처리

```
class MainHandler extends Handler {  
  
    @Override  
    public void handleMessage(Message msg) {  
        super.handleMessage(msg);  
  
        Bundle bundle = msg.getData();  
        int value = bundle.getInt("value");  
        textView.setText("value 값: " + value);  
    }  
}
```

② 핸들러 안에서 전달받은 메시지 객체 처리하기



핸들러 사용하기

- 스레드에서 핸들러로 전달
- 앱 실행하면 정상 동작

```
class BackgroundThread extends Thread {  
    int value = 0;  
  
    public void run() {  
        for (int i = 0; i < 100; i++) {  
            try {  
                Thread.sleep(1000);  
            } catch (Exception e) {}  
            value += 1;  
            Log.d("Thread", "value : " + value);  
  
            Message message = handler.obtainMessage();  
            Bundle bundle = new Bundle();  
            bundle.putInt("value", value);  
            message.setData(bundle);  
  
            handler.sendMessage(message); —→ ① 핸들러로 메시지 객체 보내기  
        }  
    }  
}
```





Runnable 객체 실행하기

- 핸들러 클래스

- 메시지 전송 방법 이외에 Runnable 객체를 실행시킬 수 있는 방법을 제공함

- Runnable 객체

- 새로 만든 Runnable 객체를 핸들러의 post() 메소드를 이용해 전달해 주기만 하면 이 객체에 정의된 run() 메소드 내의 코드들은 메인 스레드에서 실행됨



Runnable 객체 실행하기

- Runnable 객체 안에서 처리하면 메인 스레드에서 실행되게 됨

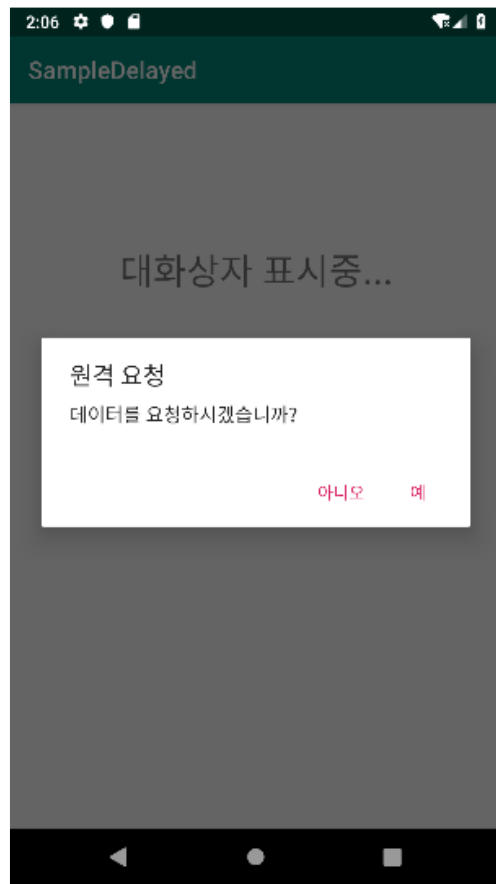
```
class BackgroundThread extends Thread {  
    int value = 0;  
  
    public void run() {  
        for (int i = 0; i < 100; i++) {  
            try {  
                Thread.sleep(1000);  
            } catch (Exception e) {}  
  
            value += 1;  
            Log.d("Thread", "value : " + value);  
  
            handler.post(new Runnable() { —————> ② 핸들러의 post() 메서드 호출하기  
                @Override  
                public void run() {  
                    textView.setText("value 값: " + value);  
                }  
            });  
        }  
    }  
}
```

2.

일정시간 후에 실행하기



일정시간 후에 실행하기 - 가상의 네트워크 요청



[버튼을 클릭했을 때 가상으로 원격 서버 요청]



메인 액티비티 코드 만들기

```
private AlertDialog makeRequestDialog(CharSequence title, CharSequence message,
                                     CharSequence titleButtonYes, CharSequence titleButtonNo) {
    AlertDialog.Builder requestDialog = new AlertDialog.Builder(this);
    requestDialog.setTitle(title);
    requestDialog.setMessage(message);
    requestDialog.setPositiveButton(titleButtonYes, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialogInterface, int i) {
            textView.setText("5초 후에 결과 표시됨.");

            handler.postDelayed(new Runnable() { —→ 핸들러의 postDelayed() 메서드 호출하기
                @Override
                public void run() {
                    textView.setText("요청 완료됨.");
                }
            }, 5000);
        }
    });
}
```



일정시간 후에 실행하기 - 핸들러 사용하기

- 네트워킹을 위한 별도의 스레드를 만들지 않고 문제를 해결하는 방법

- 핸들러의 특정 메소드를 이용하여 일정 시간 후에 실행되도록 함

- 핸들러

- 핸들러는 메시지 큐를 사용하므로 메시지들을 순서대로 처리하지만 메시지를 넣을 때 시간을 지정하면 원하는 시간에 메시지를 처리하도록 만들 수 있으므로 일정 시간 후에 실행시킬 때 유용하게 사용됨.

- 시간을 지정하는 경우

- 핸들러의 `sendMessage()` 메소드와 유사한 이름을 가진 다음과 같은 두 가지 메소드를 사용할 수 있음

- API

- `public boolean sendMessageAtTime(Message msg, long uptimeMillis)`

- `public boolean sendMessageDelayed(Message msg, long delayMillis)`

- 첫 번째 메소드는 메시지를 보낼 때 시간을 지정할 수 있으며, 두 번째 메소드는 메시지가 일정 시간이 지난 후 실행되도록 설정할 수 있음
- `Runnable` 객체를 실행하는 `post()` 메소드의 경우에도 `postAtTime()`과 `postDelayed()` 메소드가 있어 동일한 기능을 수행함

3.

스레드로 메시지 전송하기



스레드로 메시지 전송하기

- **핸들러의 기능**

- 새로 만든 스레드에서 메인 스레드로 메시지를 전달하는 것임

- **스레드의 작업 결과물을 메시지로 만들어 전달하는 이유**

- 별도의 스레드에서 메인 스레드가 관리하는 UI 객체에 직접 접근할 수 없기 때문임

- **메인 스레드에서 별도의 스레드로 메시지를 전달하는 방법이 필요한 경우**

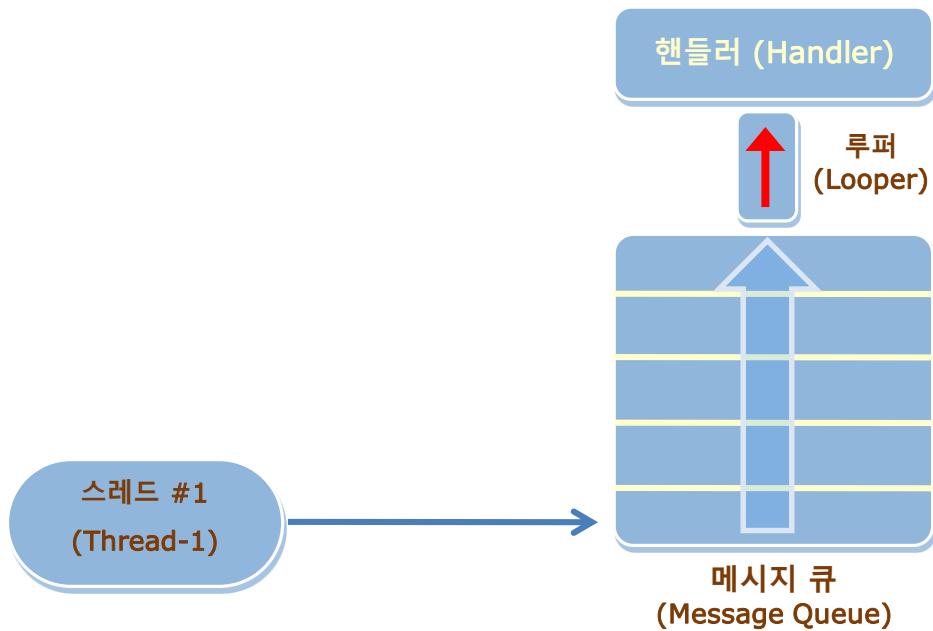
- 일반적으로 변수 선언을 통해 데이터를 전달하는 것이 가장 쉬운 방법임

- 핸들러가 사용하는 메시지 큐를 이용하여 순차적으로 메시지를 실행하는 방식이 필요한 경우가 발생함

- 특히 별도의 스레드가 동일한 객체에 접근할 때 다른 스레드들이 동시에 메소드를 호출하는 경우가 있을 수 있으므로 메시지 큐를 이용한 접근 방식에 대해 이해가 필요함



루퍼 이해하기



• 루퍼

- 무한 루프 방식을 이용해 메시지 큐에 들어오는 메시지를 지속적으로 보면서 하나씩 처리함

[루퍼를 이용한 메시지 처리]



화면 레이아웃 만들기

activity_main.xml MainActivity.java

Palette

- Common
 - Ab TextView
- Text
 - Button
- Buttons
 - ImageView
- Widgets
 - RecyclerView
- Layouts
 - <> <fragment>
- Containers
 - ScrollView
- Google
 - Switch
- Legacy

Component Tree

- ConstraintLayout
 - button- "스레드로 보내기"
 - Ab textView- "결과"
 - Ab editText(Plain Text)

Attributes

Ab textView

id: textView

Declared Attributes

Layout

Constraint Widget

Constraints (3)

- layout_width: wrap_content
- layout_height: wrap_content
- visibility:
- visibility:

Common Attributes

- text: 결과
- text:
- contentDescription:
- textAppearance: @android:style/TextAppea
- alpha:



메인 액티비티 코드 만들기

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

    editText = findViewById(R.id.editText);
    textView = findViewById(R.id.textView);

    Button button = findViewById(R.id.button);
    button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            String input = editText.getText().toString();
            Message message = Message.obtain();
            message.obj = input;

            thread.processHandler.sendMessage(message); → ❶ 새로 만든 스레드 안에 있는 핸들러로
        }                                           메시지 전송하기
    });

    thread = new ProcessThread();
}
```



메인 액티비티 코드 만들기 (계속)

```
class ProcessThread extends Thread {  
    ProcessHandler processHandler = new ProcessHandler();  
  
    public void run() {  
        Looper.prepare();  
        Looper.loop();  
    }  
}
```

```
class ProcessHandler extends Handler {
```

```
    public void handleMessage(Message msg) {  
        final String output = msg.obj + " from thread.";
```

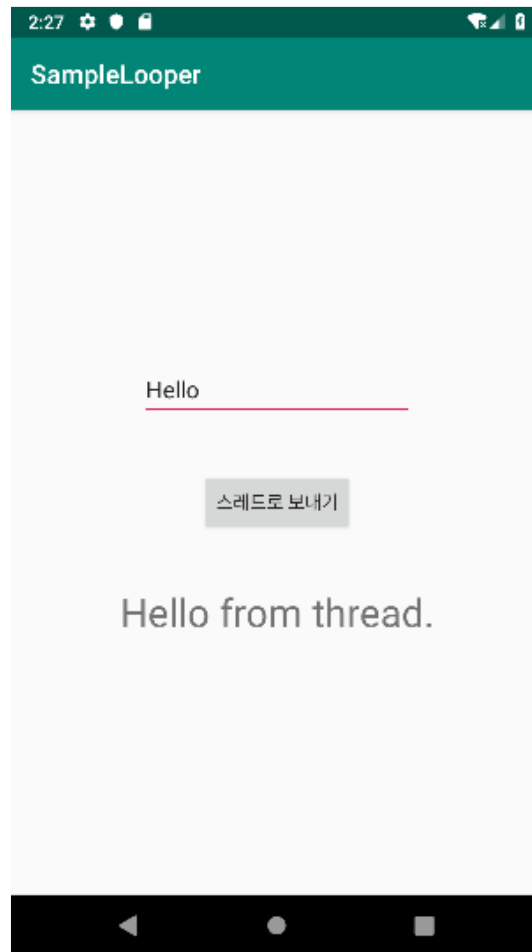


② 새로 만든 스레드 안에서 전달받은 메시지 처리하기

```
        handler.post(new Runnable() {  
            @Override  
            public void run() {  
                textView.setText(output);  
            }  
        });  
    }  
}
```



실행 화면



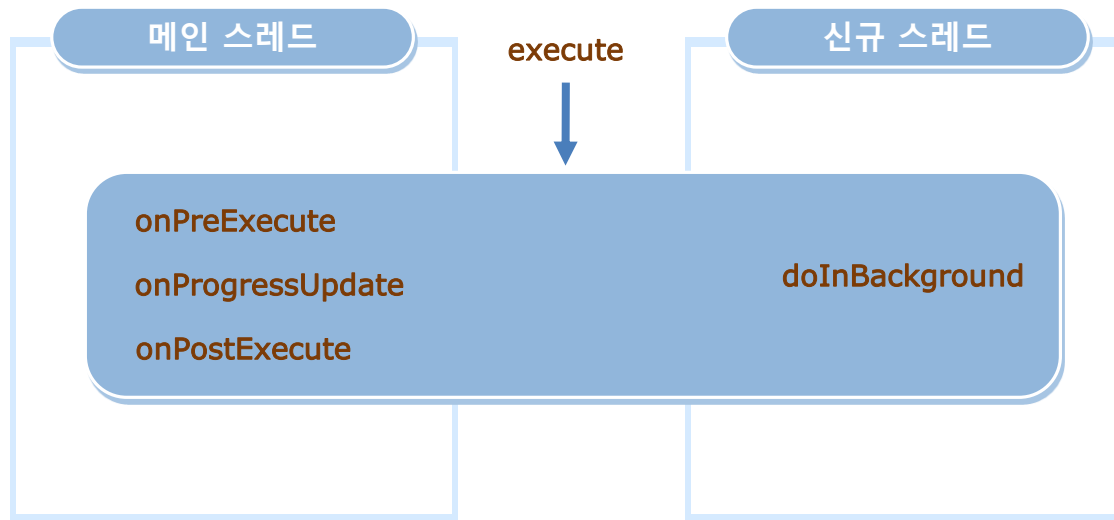
[루퍼를 이용한 스레드 간 문자열 전달]

4.

AsyncTask 사용하기



AsyncTask 사용하기



[AsyncTask를 이용한 작업 수행 방식]

• AsyncTask

- 백그라운드 작업을 좀 더 쉽고 간단하게 하고 싶다면 AsyncTask 클래스를 사용할 수 있음
- AsyncTask 객체를 만들고 execute() 메소드를 실행하면 이 객체는 정의된 백그라운드 작업을 수행하고 필요한 경우에 그 결과를 메인 스레드에서 실행하므로 UI 객체에 접근하는데 문제가 없게 됨



AsyncTask의 주요 메소드

메소드 이름	설 명
doInBackground	<ul style="list-style-type: none">- 새로 만든 스레드에서 백그라운드 작업 수행- execute() 메소드를 호출할 때 사용된 파라미터를 배열로 전달받음
onPreExecute	<ul style="list-style-type: none">- 백그라운드 작업 수행 전 호출- 메인 스레드에서 실행되며 초기화 작업에 사용
onProgressUpdate	<ul style="list-style-type: none">- 백그라운드 작업 진행 상태를 표시하기 위해 호출- 작업 수행 중간 중간에 UI 객체에 접근하는 경우 사용- 이 메소드가 호출되도록 하려면 백그라운드 작업 중간에 publishProgress() 메소드를 호출
onPostExecute	<ul style="list-style-type: none">- 백그라운드 작업이 끝난 후 호출- 메인 스레드에서 실행되며 메모리 리소스를 해제하는 등의 작업에 사용- 백그라운드 작업의 결과는 Result 타입의 파라미터로 전달

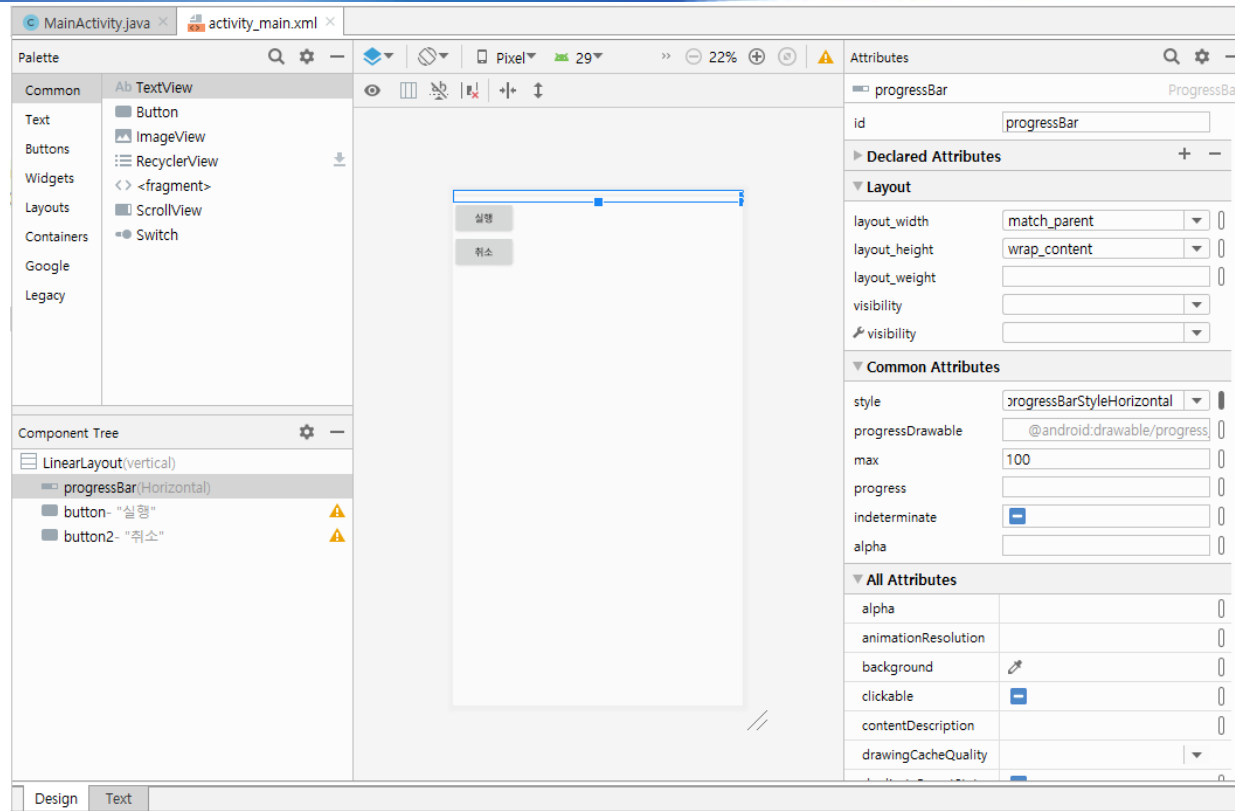


Async 메소드 사용하기

- AsyncTask 객체의 cancel() 메소드
 - 작업을 취소함, 이 메소드를 통해 작업을 취소했을 경우에는 onCancelled() 메소드가 호출됨
- AsyncTask 객체의 getStatus() 메소드
 - 작업의 진행 상황을 확인함
 - 메소드를 호출했을 때 리턴되는 AsyncTask.Status 객체는 상태를 표현함
 - 각각의 상태는 PENDING, RUNNING, FINISHED로 구분됨
- PENDING
 - 작업이 아직 시작되지 않았다는 것을 의미함
- RUNNING
 - 실행 중임을 의미함
- FINISHED
 - 종료되었음을 의미함



화면 레이아웃 구성하기



[AsyncTask를 이용해 진행 상태를 표시하는 화면 구성]

- 이 화면의 버튼을 클릭하면 별도의 스레드에서 값을 1씩 증가시키도록 함
- 100밀리 초마다 한 번씩 값을 증가시키므로 프로그레스바의 최대값으로 지정된 100이 될 때까지 10초가 걸리게 됨



메인 액티비티 코드 만들기

- 백그라운드 작업을 수행할 클래스는 BackgroundTask라는 이름의 클래스로 정의하고 AsyncTask 클래스를 상속받음
- onPreExecute() 메소드는 초기화 단계에서 사용되므로 값을 저장하기 위해 메인 액티비티에 정의한 value 변수의 값을 0으로 초기화하고 프로그레스바의 값도 0으로 만들어 줌
- doInBackground() 메소드는 주 작업을 실행하는데 사용되므로 while 구문을 이용해 value의 값을 하나씩 증가시키도록 함

```
Button button = findViewById(R.id.button);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        task = new BackgroundTask();
        task.execute();
    }
});
```

➔ ① 태스크 객체 만들어 실행하기



AsyncTask에서 상속한 클래스 정의

```
class BackgroundTask extends AsyncTask<Integer , Integer , Integer> {  
    protected void onPreExecute() {  
        value = 0;  
        progressBar.setProgress(value);  
    }  
  
    protected Integer doInBackground(Integer ... values) {  
        while (isCancelled() == false) {  
            value++;  
            if (value >= 100) {  
                break;  
            } else {  
                publishProgress(value);  
            }  
  
            try {  
                Thread.sleep(100);  
            } catch (InterruptedException ex) {}  
        }  
  
        return value;  
    }  
}
```

② 태스크 객체 안에서 백그라운드 작업 수행하도록 하기

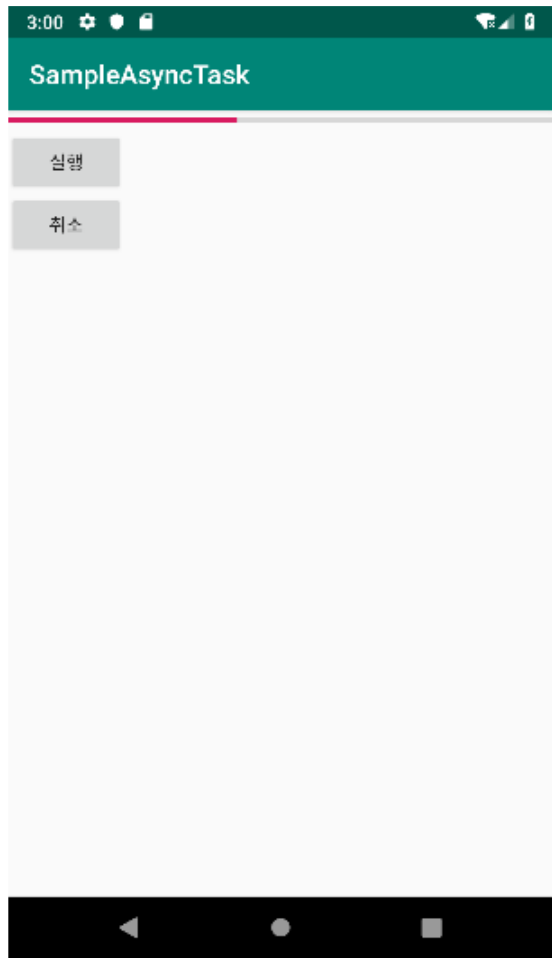
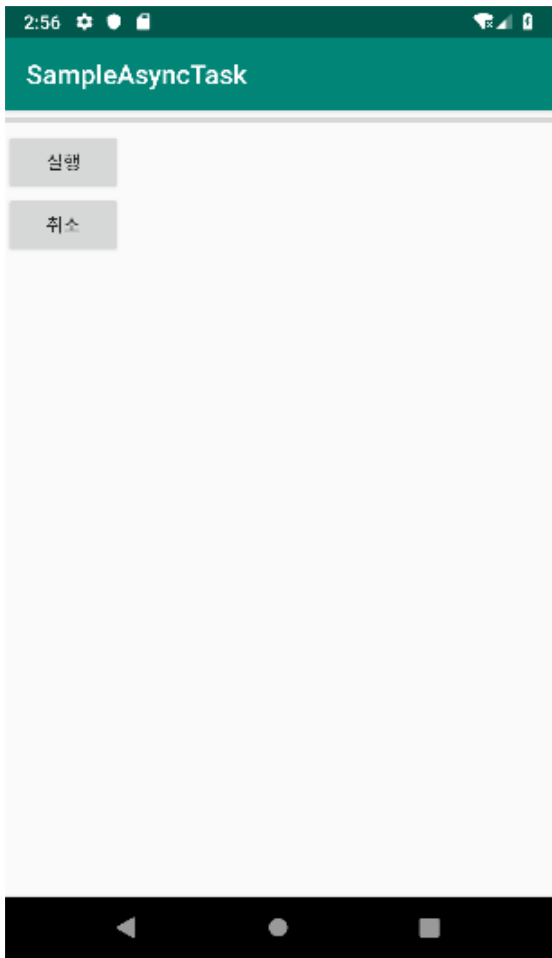


AsyncTask에서 상속한 클래스 정의 (계속)

```
protected void onProgressUpdate(Integer ... values) {  
    progressBar.setProgress(values[0].intValue());  
}  
  
protected void onPostExecute(Integer result) {  
    progressBar.setProgress(0);  
}  
  
protected void onCancelled() {  
    progressBar.setProgress(0);  
}  
}  
}
```



앱 실행

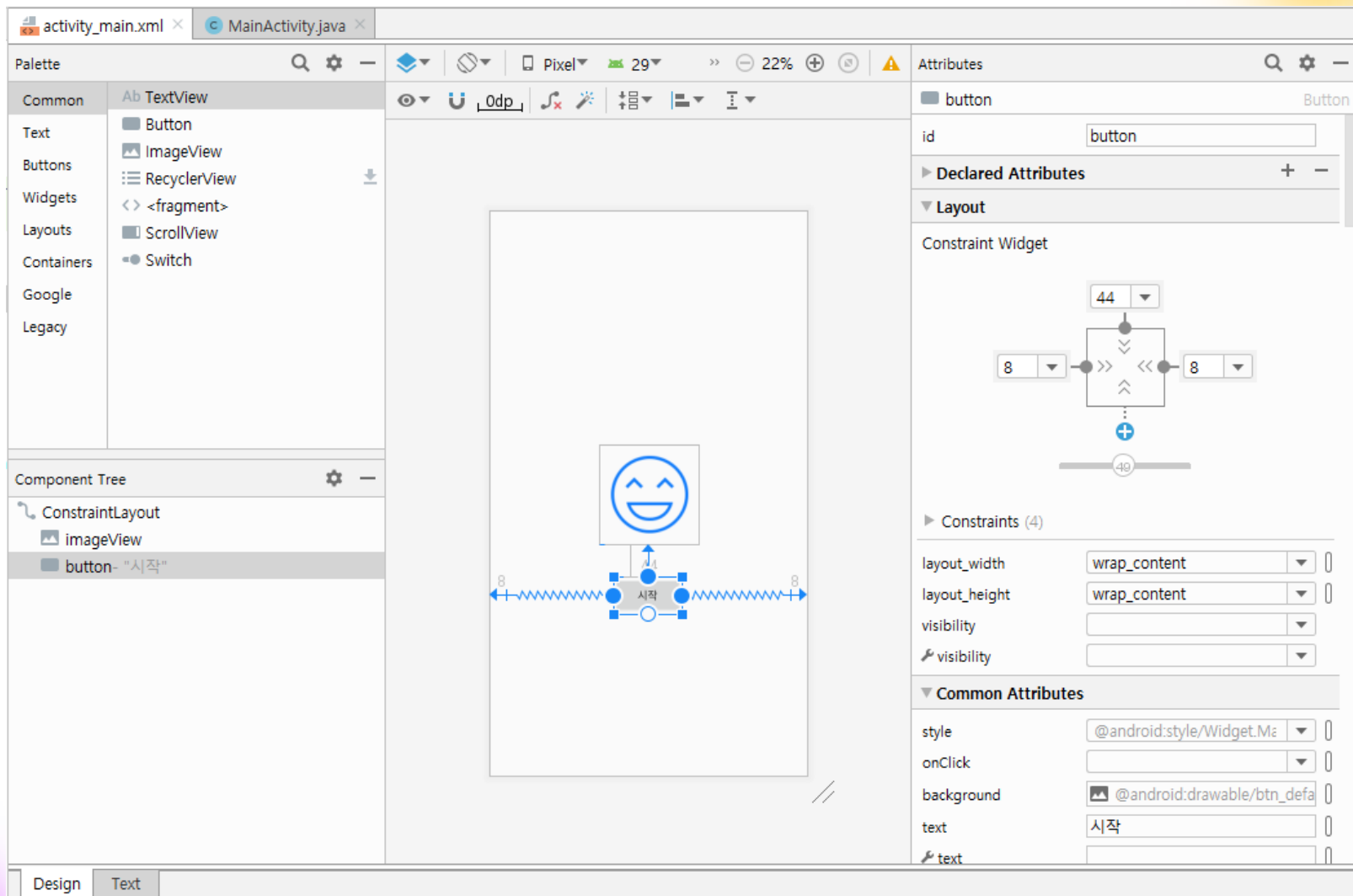


5.

스레드로 애니메이션 만들기



레이아웃 화면 만들기





스레드 정의

```
class AnimThread extends Thread {  
    public void run() {  
        int index = 0;  
        for (int i = 0; i < 100; i++) {  
            final Drawable drawable = drawableList.get(index);  
            index += 1;  
            if (index > 4) {  
                index = 0;  
            }  
        }  
    }  
}
```

```
handler.post(new Runnable() {  
    @Override  
    public void run() {  
        imageView.setImageDrawable(drawable);  
    }  
});  
  
try {  
    Thread.sleep(1000);  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}  
}  
}  
}
```

② 화면에 이미지를 보여주기 위해 핸들러의 post() 메서드 호출하기

