모바일 프로그래밍

11 프로세스와 스레드 2

2017 2학기

강승우

작업 스레드에서 UI 변경 방법

작업 스레드에서 UI 변경 방법 1

- Handler 객체의 post 함수 이용하는 방법
  - Activity 멤버 변수로 Handler 객체를 생성한다
  - 작업 Thread에서 UI 화면 갱신이 필요한 경우,
    UI 화면을 갱신하는 작업을 Runnable 인터페이스의 run() 함수로 구현하고 이를 Handler의 post 함수의 인자로 전달한다
    - 이 post 함수는 메인 스레드에서 처리할 작업을 넣어두는 메시지 큐에 새로운 작업 메시지를 넣는 역할을 함
  - 메인 스레드가 큐에서 작업 메시지를 꺼내서 처리하면 UI 화면이 갱신됨

  - 예제 프로젝트 이름: 10_Process_Thread\Ch13Handler

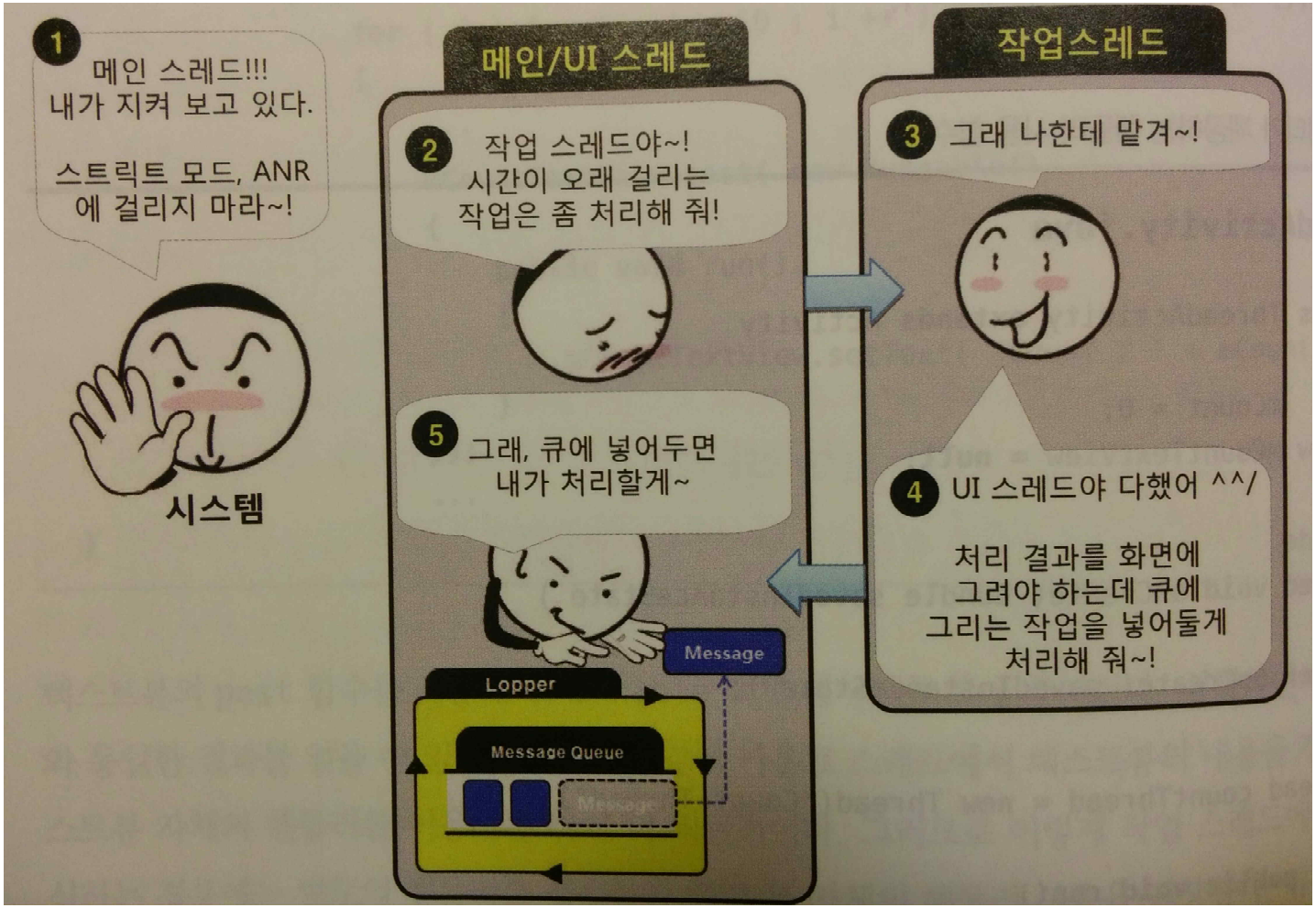
Code snippet

```
Handler mHandler = new Handler();

class CountThread extends Thread {
    public void run() {
        int i = 0;
        for(i=0; i < 20 && running; i++) {
            mCount++;
            //tv.setText("Count: " + mCount);

            mHandler.post(new Runnable() {
                @Override
                public void run() {
                    tv.setText("Count: " + mCount);
                }
            });
        …
        …
}
```

작업 스레드에서 UI 상의 뷰를 갱신하는 작업을 할 수 없음

UI 갱신 작업을 Runnable 인터페이스의 run() 함수 내에 구현하고 Handler 객체의 post 함수로 넘겨준다

메인 스레드와 작업 스레드 관계



작업 스레드에서 UI 변경 방법 2

- View의 post, postDelayed 함수 이용
  - 별도의 Handler 객체를 생성하지 않음
  - View에는 자체적으로 Handler를 포함하고 있으므로 post 함수를 바로 이용할 수 있음

```
public void run() {
    int i = 0;
    for(i=0; i < 20 && running; i++) {
        mCount++;

        tv.post(new Runnable() {
            @Override
            public void run() {
                tv.setText("Count: " + mCount);
            }
        });
```

- 작업 스레드에서 특정 뷰를 갱신하는 경우 별도의 핸들러를 생성하지 않고 뷰 자체의 핸들러 사용을 권장

View.post(Runnable action)
View.postDelayed(Runnable action, long delayMillis)

작업 스레드에서 UI 변경 방법 3

- Activity 클래스의 runOnUiThread 함수 이용
  - MainActivity.this.runOnUiThread(Runnable action);
  - runOnUiThread 함수를 이용하여 Runnable 객체 메시지를 큐에 추가
  - 특정 뷰 하나가 아닌 액티비티에서 사용되는 여러 뷰를 동시에 갱신하고자 할 때 사용하는 것이 직관적임

```
public void run() {
    int i = 0;
    for(i=0; i < 20 && running; i++) {
        mCount++;
        MainActivity.this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                tv.setText("Count: " + mCount);
            }
        });
```

작업 스레드 관련 클래스

AsyncTask 클래스

- AsyncTask
  - 작업 스레드 이용과 관련한 복잡한 부분을 쉽게 처리할 수 있도록 도와주는 추상 클래스
  - 안드로이드에서 요구하는 메인 스레드와 작업 스레드의 분리 구조를 쉽게 구현할 수 있게 해줌

- AsyncTask 클래스의 오버라이드 가능한 5가지 재정의 함수
  - onPreExecute()
  - doInBackground(Params…): 추상 함수로 반드시 구현되어야 함
  - onProgressUpdate(Progress…)
  - onCancelled()
  - onPostExecute(Result)

https://developer.android.com/reference/android/os/AsyncTask.html

AsyncTask

- AsyncTask는 3개의 generic type을 제공
- AsyncTask<Params, Progress, Result>
  - Params
    - 실행할 때 전달할 인수의 타입. execute() 함수 호출 시 입력 파라미터로 넘겨주는 값의 타입을 지정
    - doInBackground 함수의 입력 파라미터로 전달
  - Progress
    - 매 작업 단계마다 진행 상태를 표기하는 타입
    - publishProgress() 함수 호출 시 입력 파라미터로 넘겨주는 값의 타입을 지정
  - Result
    - 작업 결과로 리턴될 타입. doInBackground 함수에서 return 하는 값의 타입을 지정
    - onPostExecute 함수의 입력 파라미터로 전달
  - 미사용 타입에 대해서는 Void로 지정

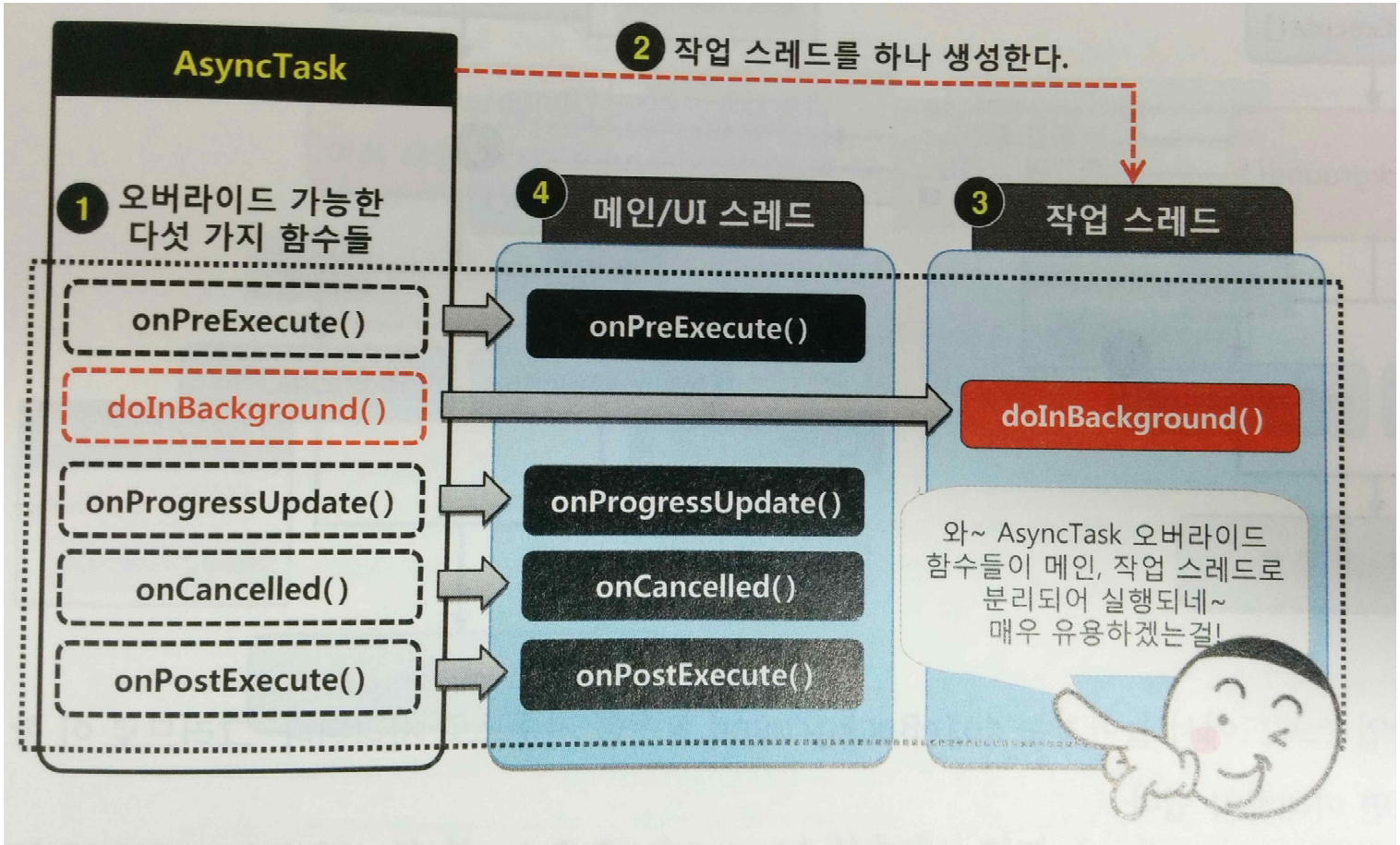
AsyncTask의 동작

하나의 작업 스레드를 구동하고 **5가지 재정의 함수**를 **메인 스레드와 작업 스레드로 분리**하여 실행

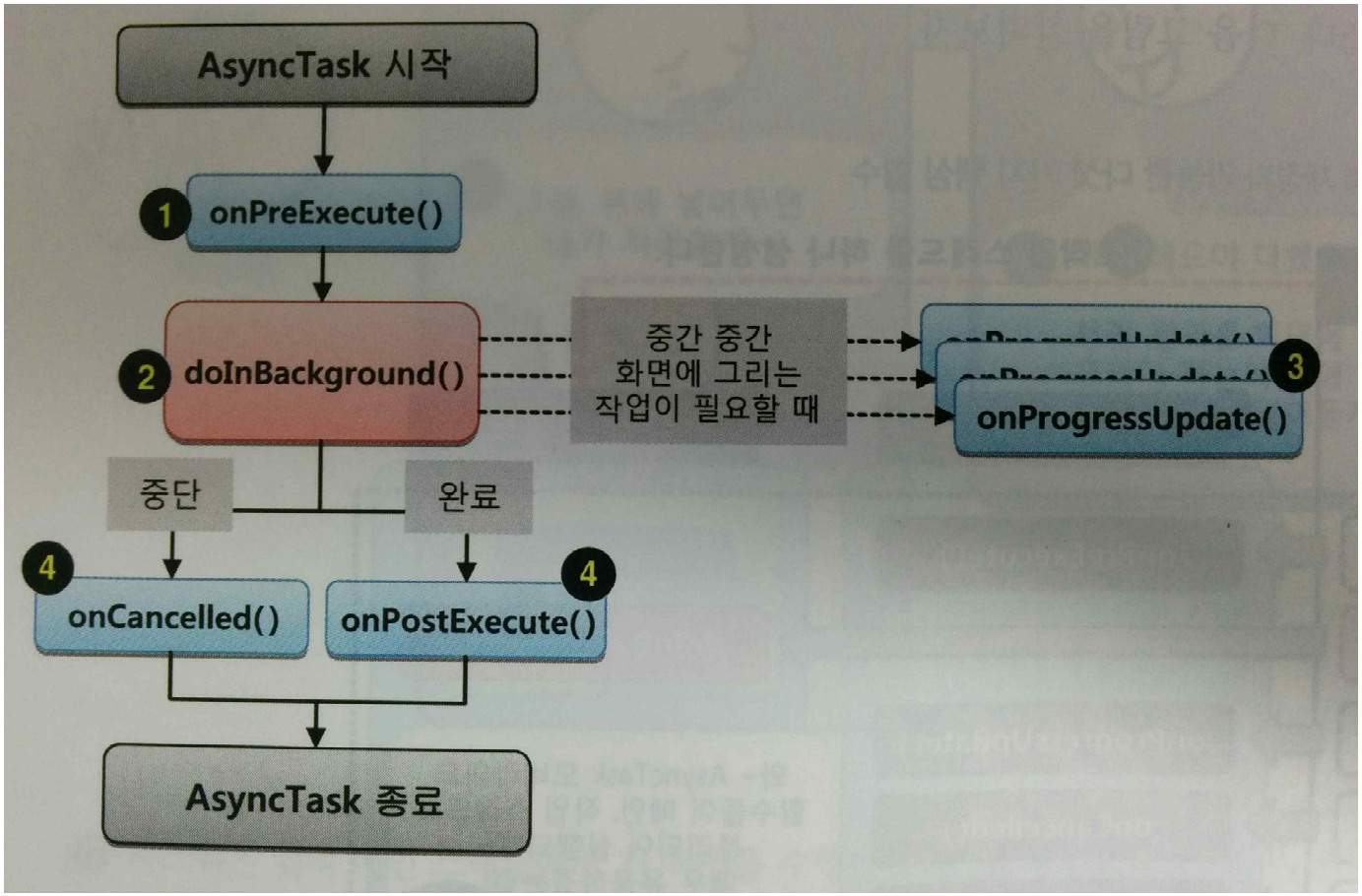
- AsyncTask는 내부적으로 작업 스레드를 하나 생성하고 실행함
- AsyncTask가 생성한 스레드에서는 재정의 함수인 doInBackground 함수를 실행함
- 나머지 4가지 재정의 함수는 메인 스레드에서 실행함

→ 오래 걸리는 일을 처리하려고 할 때는 AsyncTask를 상속받아서 5개의 함수를 재정의하면 된다

→ 오래 걸리는 작업은 doInBackground 함수로 구현하고 UI 화면을 갱신하는 일은 나머지 함수로 구현한다



AsyncTask의 생명주기



1. 가장 먼저 **onPreExecute** 함수 실행
doInBackground 함수가 호출되기 전 화면 갱신 작업을 처리하는 용도로 사용

2. **doInBackground** 함수 실행
오래 걸리는 작업을 처리 (작업 스레드)

3. **onProgressUpdate** 함수의 실행은
doInBackground 함수 처리 도중 화면을 갱신하는 작업이 필요할 때
publishProgress 함수를 호출한 경우 이루어짐
doInBackground 함수의 진행 정도를 표시하는 용도로 사용

4. doInBackground 함수 처리도중 AsyncTask를 중단하면 **onCancelled** 함수 실행, 처리가 완료되면 **onPostExecute** 함수 실행. 작업 스레드가 종료됨을 의미

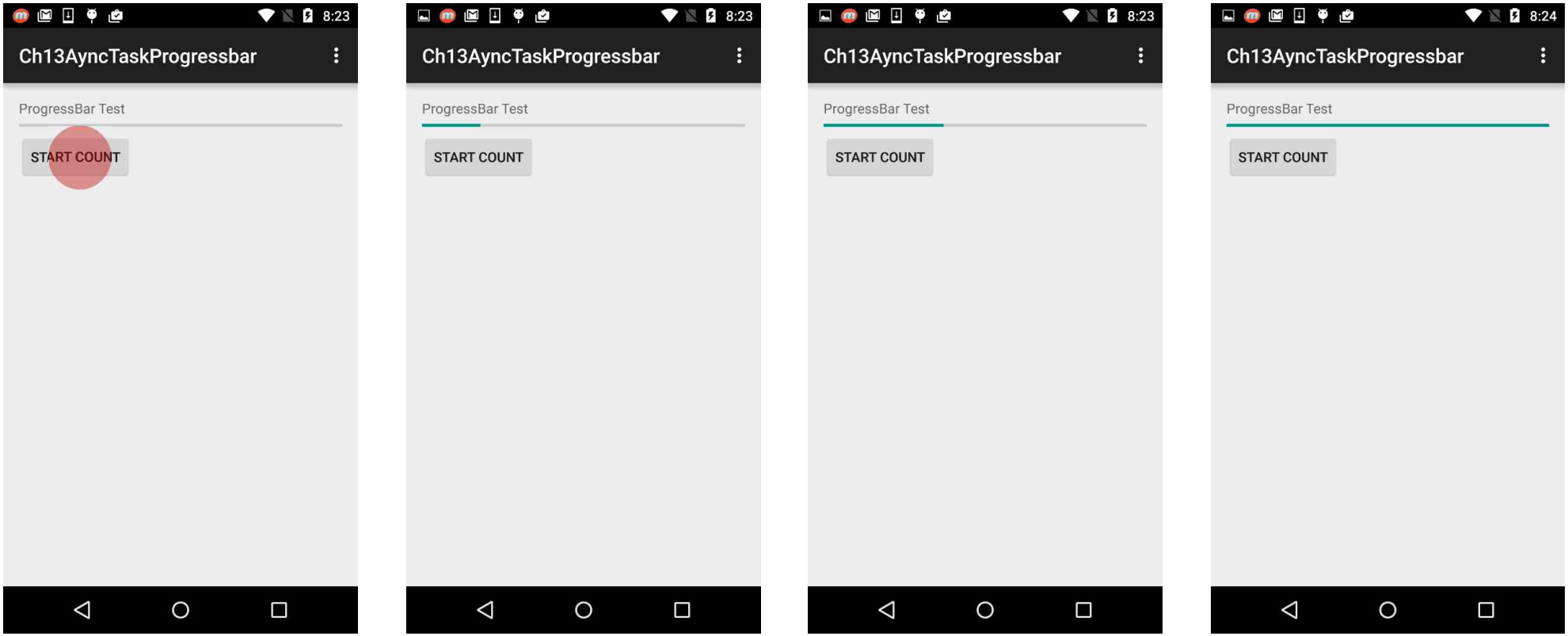
AsyncTask의 상태

- Status.PENDING
  - AyncTask 객체만 생성하고 execute 등의 함수로 실행하지 않은 상태
- Status.RUNNING
  - 실행 중인 상태
- Status.FINISHED
  - 종료 상태
- getStatus()
  - 현재 상태 얻기

- AsyncTask 실행 중 실행 취소
  - AsyncTask 클래스의 cancel() 함수 호출
  - 예: task.cancel(true);
  - true이면 task를 실행중인 스레드는 interrupt 됨
    - InterruptedException에서 걸려서 중단
- 더 유연한 중단 방법
  - isCancelled() 함수 사용
  - doInBackground 함수 내에서 isCancelled() 결과가 true인지 검사하여 종료할 수 있음

AsyncTask 이용 예제 1

- 프로그레스 바의 진행 상태를 AsyncTask를 이용하여 표시
- 예제 프로젝트 이름: 10_Process_Thread\Ch13AsyncTaskProgressbar



주요 code snippet

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

    mProgress = (ProgressBar)findViewById(R.id.progressbar);
    Button btn = (Button)findViewById(R.id.button);
    btn.setOnClickListener(new Button.OnClickListener() {
        public void onClick(View v) {
            new CounterTask().execute();
        }
    });
}
```

CounterTask 객체를 생성하고 execute() 함수 호출한다

```
class CounterTask extends AsyncTask<Void, Integer, Integer> {
    protected void onPreExecute() {
    }

    protected Integer doInBackground(Void... params) {
        while(mProgressStatus < 100) {
            try {
                // 백그라운드 작업에 시간이 소요되는 과정을 가정
                Thread.sleep(500);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            mProgressStatus++;
            publishProgress(mProgressStatus);
        }
        return mProgressStatus;
    }

    protected void onProgressUpdate(Integer... value) {
        mProgress.setProgress(value[0]);
    }

    protected void onPostExecute(Integer result) {
        mProgress.setProgress(result);
        mProgressStatus = 0;
    }
}
```

Integer 데이터 하나가 전달됨

AsyncTask 이용 예제 2

- 서버에 존재하는 파일을 다운로드 받아 그 과정을 표시
  - 네트워크 연결 및 다운로드 작업은 실제 하지 않고 가상으로 된다고 가정하여 구현한 예제

- 예제 프로젝트 이름
  - 10_Process_Thread\Ch13AsyncTaskDownload

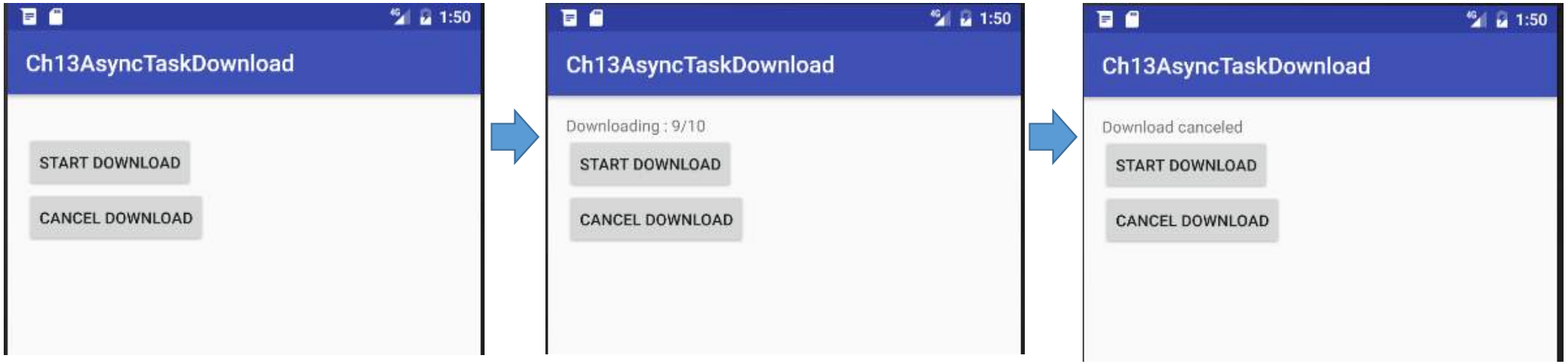
AsyncTask 이용 예제 2

Start download 버튼 클릭 시

AsyncTask 실행되면서 downloading 상태 업데이트

Cancel download 버튼 클릭 시

AsyncTask 실행 취소 onCancelled() 메소드 실행



Code snippet

버튼 클릭 이벤트 처리를 위한 콜백 메소드

```
// 시작(start download) 버튼 눌렀을 때 호출
public void onClickStart(View v) {
    // 만일 파일 내려받기 AsyncTask 객체가 null이면 객체를 생성하고 실행
    if(mFileDownloadTask == null) {
        // 파일 내려받기 AsyncTask 객체를 생성하고 실행
        mFileDownloadTask = new FileDownloadTask();
        mFileDownloadTask.execute("FileUrl_1", "FileUrl_2", "FileUrl_3", "FileUrl_4", "FileUrl_5",
                "FileUrl_6", "FileUrl_7", "FileUrl_8", "FileUrl_9", "FileUrl_10");
    }
}

// 취소(cancel download) 버튼 눌렀을 때 호출
public void onClickCancel(View v) {
    // 만일 파일 내려받기 AsyncTask가 종료 상태가 아니면 진행을 취소
    if(mFileDownloadTask != null && mFileDownloadTask.getStatus() != AsyncTask.Status.FINISHED) {
        mFileDownloadTask.cancel(false);
        mFileDownloadTask = null;
    }
}
```

```
class FileDownloadTask extends AsyncTask<String, Integer, Boolean> {
    // AsyncTask 작업 시작시 화면 갱신을 위한 메소드
    @Override
    protected void onPreExecute() {
        // 화면에 최초 내려받기 시도를 알리는 텍스트 출력
        mState.setText("File Download ...");
    }

    // 작업 스레드에 의해 처리되는 루틴을 정의하는 메소드
    @Override
    protected Boolean doInBackground(String... downloadInfos) {
        int totalCount = downloadInfos.length;

        // 전달받은 파일 Url 개수만큼 반복하면서 파일을 내려받는다
        for(int i = 0; i < totalCount; i++) {
            // 1. 파일 내려받기 처리 상태를 화면에 표시하기 위해 호출
            // 두 개의 integer 값을 파라미터로 전달
            publishProgress(i+1, totalCount);

            // 아래 isCancelled()로 취소 여부 확인하는
            // 루틴이 있는 경우와 없는 경우 차이를 확인해보자
            //if(isCancelled() == true) {
            //    return false;
            //}

            // 2. 아래를 파일을 내려받는 과정이라고 가정
            try { Thread.sleep(500); }
            catch (InterruptedException e) {e.printStackTrace(); return false;}
        }
        return true;
    }
```

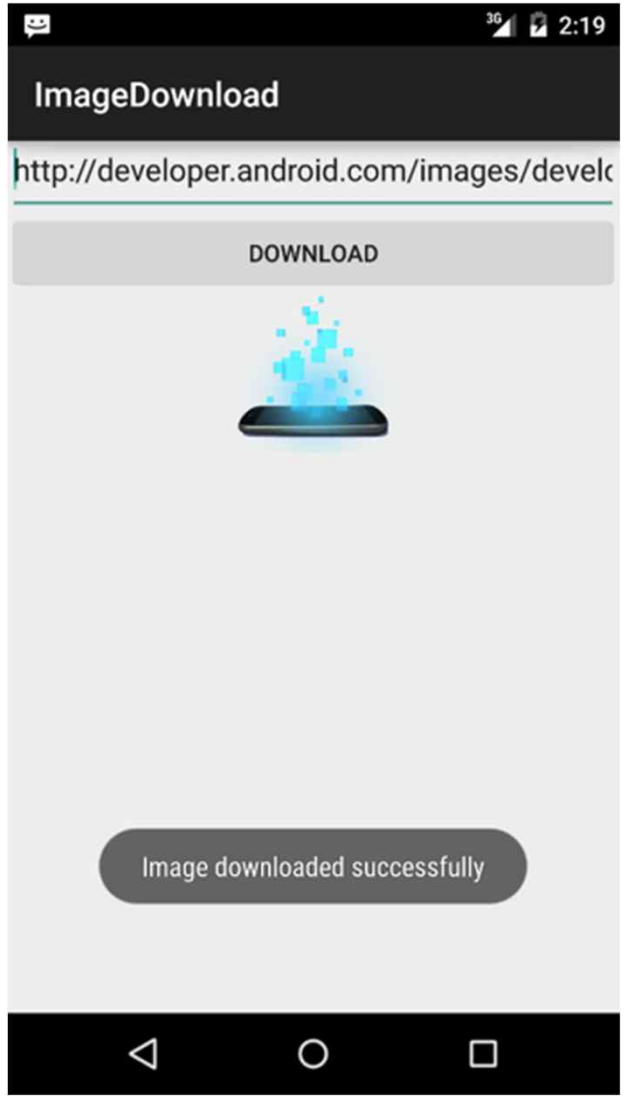
```
    @Override
    protected void onProgressUpdate(Integer... downloadInfos) {
        // publishProgress 함수에서 2개의 integer 파라미터를 전달
        // downloadInfos는 2개의 integer를 담고 있는 배열로 접근 가능
        int currentCount = downloadInfos[0];
        int totalCount = downloadInfos[1];

        // 화면에 현재의 파일 내려받기 상태를 표시. 예) Downloading: 3/10
        mState.setText("Downloading : " + currentCount + "/" + totalCount);
    }

    // cancel() 함수 호출 시 실행되는 메소드
    @Override
    protected void onCancelled() {
        // 화면에 내려받기가 취소되었음을 표시
        mState.setText("Download canceled");
    }

    // doInBackground() 함수 실행 종료 시 실행되는 메소드
    @Override
    protected void onPostExecute(Boolean result) {
        // 화면에 내려받기 성공/실패 여부를 텍스트로 출력
        if (true == result) {
            mState.setText("Download finished");
        } else {
            mState.setText("Download failed");
        }
        mFileDownloadTask = null;
    }
}
```

AsyncTask 예제 3

- 입력된 URL로부터 이미지를 다운로드 하여 화면 상의 이미지 뷰에 표시

- 예제 프로젝트 이름
  - 10_Process_Thread\Ch13ImageAsyncTask
- 설명
  - Download 버튼을 누르면 EditText 상에 입력된 http 주소에 연결하여 이미지 파일을 다운로드 받고 완료가 되면 화면 상의 ImageView에 표시

- AndroidManifest.xml 파일에 permission 설정 필요
- <uses-permission android:name="android.permission.INTERNET"/>



```
public void onClick(View view) {
    EditText url = (EditText)findViewById(R.id.et_url);
    DownloadTask task = new DownloadTask();
    task.execute(url.getText().toString());
}

class DownloadTask extends AsyncTask<String, Void, Bitmap> {
    Bitmap bitmap = null;

    @Override
    protected Bitmap doInBackground(String... url) {
        try {
            bitmap = downloadUrl(url[0]);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return bitmap;
    }

    @Override
    protected void onPostExecute(Bitmap result) {
        ImageView imageView = (ImageView)findViewById(R.id.imageView);
        imageView.setImageBitmap(result);
        Toast.makeText(getApplicationContext(), "Image downloaded successfully", Toast.LENGTH_LONG).show();
    }
}
```

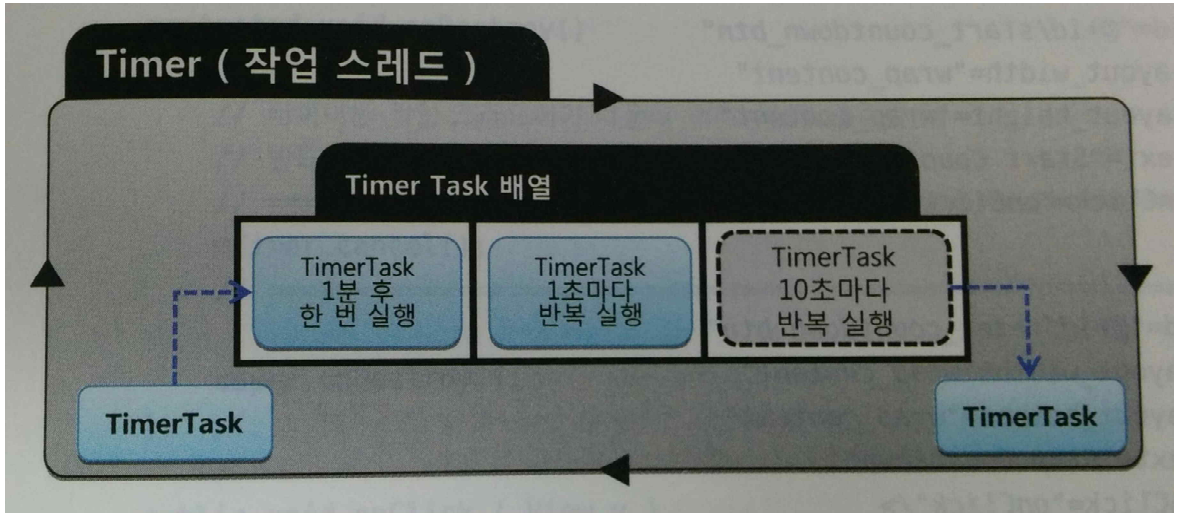
버튼을 클릭하면 AsyncTask 객체 생성 후 실행

URL 연결 후 다운로드 하는 작업 수행

다운로드 완료 후 ImageView에 표시

Timer와 TimerTask

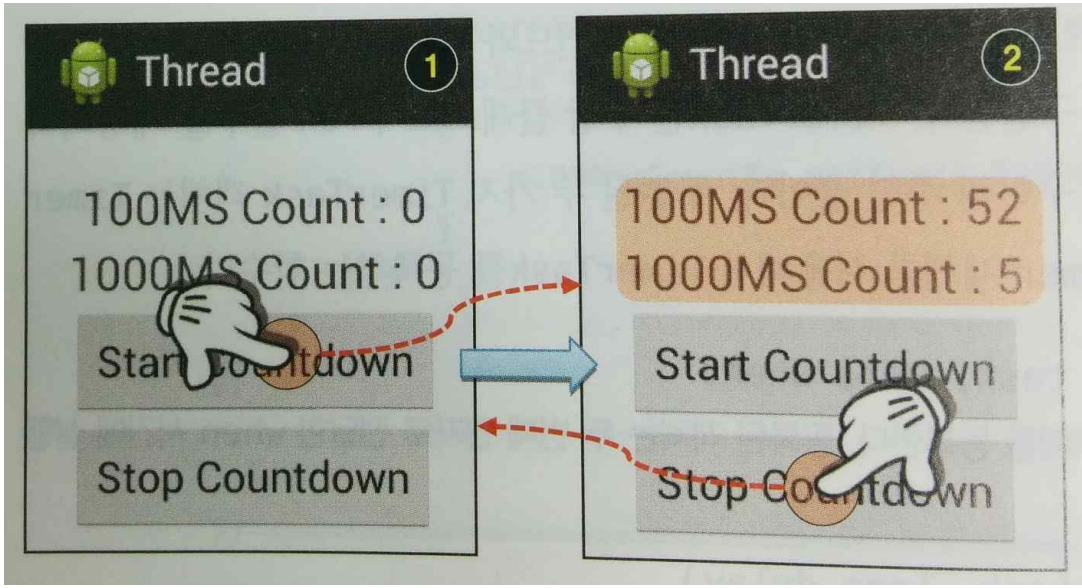
- 특정 작업을 주기적으로 실행하여야 할 때 사용
- Timer
  - 작업을 주기적으로 실행하는 클래스
  - 내부적으로 여러 개의 TimerTask를 배열로 관리하고, 해당하는 주기마다 TimerTask를 실행한다
- TimerTask
  - Timer에서 주기적으로 실행하는 작업을 정의하는 클래스



Timer와 TimerTask의 관계

Timer, TimerTask 예제

- 100 밀리초와 1000 밀리초마다 수치를 증가시키는 기능
- 예제 프로젝트 이름
  - 10_Process_Thread\Ch13TimerTask



Timer, TimerTask 예제

- TimerTask 정의
  - TimerTask를 사용하기 위해서는 run() 함수를 재정의하여야 함
  - run() 함수 안에 처리할 작업을 구현한다
  - 이 예제에서는 100 밀리초마다 처리할 내용을 담은 TimerTask 객체와 1000 밀리초마다 처리할 내용을 담은 TimerTask 객체 2개를 구현
- Timer 객체를 통해 TimerTask 객체 등록
  - mTimer.schedule(TimerTask task, long delay, long period)
    - delay 지연시간 이후에 period 간격으로 task 실행
  - mTimer.schedule(TimerTask task, long delay)
    - delay 지연시간 이후 task 실행
  - mTimer.schedule(TimerTask task, Date when)
  - mTimer.schedule(TimerTask task, Date when, long period)
- 등록된 TimerTask를 개별적으로 제거
  - TimerTask의 cancel() 함수 호출