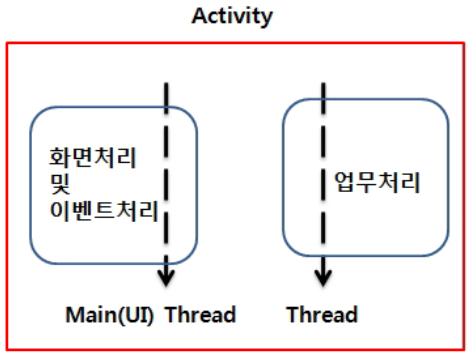
# 16장. 스레드와 핸들러

# 16.1 ANR과 스레드-핸들러

## 16.1.1. 액티비티 ANR

- 액티비티가 사용자 이벤트에 반응하지 못하는 상황

- ANR 문제 해결방법
- 스레드로 간단하게 해결




## 16.1.2. ANR 문제 해결방법

- 스레드를 만드는 방법은 Thread 클래스를 상속받아 작성하는 방법이 있고 Runnable 인터페이스를 구현하여 작성하는 방법이 있다.

```
class MyThread extends Thread {
   public void run() {
   }
}
```

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

# 16.1 ANR과 스레드-핸들러

```
class MyThread implements Runnable {
    public void run() {
    }
}
```

```
MyThread runnable = new MyThread();
Thread thread = new Thread(runnable);
thread.start();
```

- sleep ( ) 함수는 실행 중인 스레드를 Sleep Pool로 보내어 지정된 시간 동안 대기 상태가 되게 하는 방법

```
public void run() {
    int sum=0;
    for (int i = 1; i <= 10; i++) {
        sum += i;
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
        }
    }
}
```

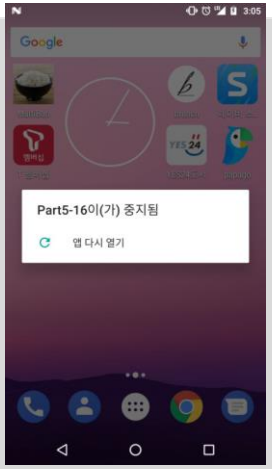
- wait ( ) 함수에 의해 대기 상 태가 된 스레드는 스스로 수행 상태가 될 수 없으며 다른 스레드에서 notify ( ) 함수를 이용해서 깨워줘 야 다시 동작

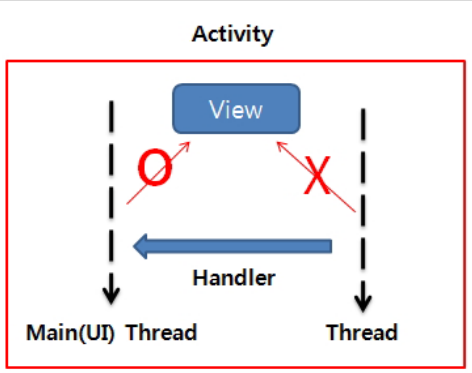
# 16.1 ANR과 스레드-핸들러

## 16.1.3. 핸들러

- **스레드-핸들러 구조**
- UI 스레드가 아닌, 개발자 스레드에서 런타임 때 액티비티의 화면 출력 요소인 뷰에 접근하면 에러가 발생

```
class MyThread extends Thread {
    public void run() {
        int sum = 0;
        for (int i = 1; i < 11; i++) {
            sum += i;
            textView.setText("sum:" + sum);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
        }
    }}
```



# 16.1 ANR과 스레드-핸들러

- 핸들러에 작업 의뢰
- post ( ) 함수를 이용

```
Handler handler=new Handler();
```

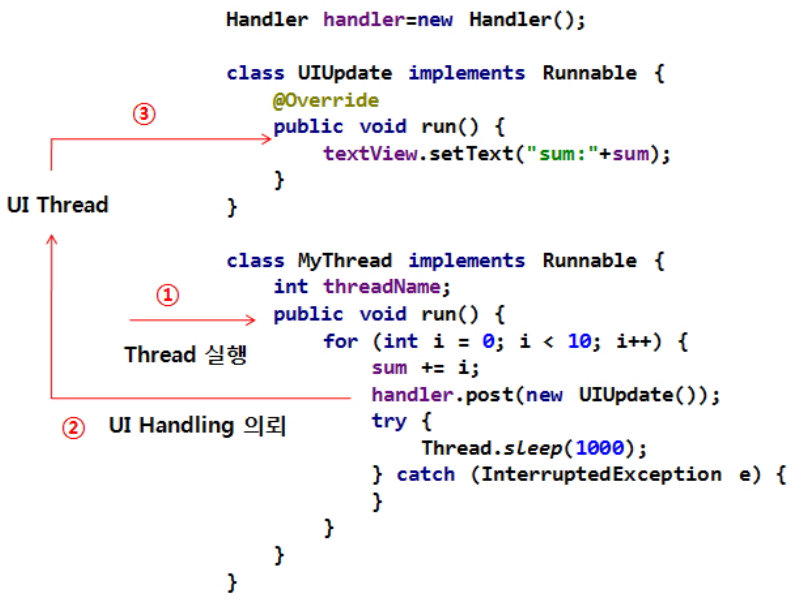
- 뷰와 관련된 작업을 담당하 는 Runnable을 구현한 클래스를 준비

```
class UIUpdate implements Runnable {
    @Override
    public void run() {
        textView.setText("sum:"+sum);
    }
}
```

- Handler의 post ( ) 함수를 호출해 UI 스레드에게 작업을 의뢰

```
class MyThread implements Runnable {
    int threadName;
    public void run() {
        for (int i = 0; i < 10; i++) {
            sum += i;
            handler.post(new UIUpdate());
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
        }
    }
}
```

# 16.1 ANR과 스레드-핸들러




```
Handler handler=new Handler();

class UIUpdate implements Runnable {
    @Override
    public void run() {
        textView.setText("sum:"+sum);
    }
}

class MyThread implements Runnable {
    int threadName;
    public void run() {
        for (int i = 0; i < 10; i++) {
            sum += i;
            handler.post(new UIUpdate());
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
            }
        }
    }
}
```

sendMessage ( ) 함수를 이용

- sendMessage(Message msg): UI 스레드에 의뢰
- sendMessageAtFrontOfQueue(Message msg): 이번 의뢰를 가장 먼저 처리
- sendMessageAtTime(Message msg, long uptimeMillis): 지정된 시간에 수행해 달라는 요청
- sendMessageDelayed(Message msg, long delayMillis): 의뢰를 바로 처리하지 말고 지정 시간 후에 수행해 달라는 요청
- sendEmptyMessage(int what): 데이터 전달 없이 의뢰하는 경우

# 16.1 ANR과 스레드-핸들러

Message 객체는 뷰작 업을 의뢰할 때 UI 스레드에 넘기는 데이터를 담는 객체

- what: int 형 변수로 구분자. 개발자가 임의의 숫자로 요청을 구분하기 위해 사용
- obj: UI 스레드에 넘길 데이터. Object 타입의 변수
- arg1, arg2: UI 스레드에 넘길 데이터, int 형

```
class MyThread extends Thread {
    public void run() {
        try {
            while (loopFlag) {
                Thread.sleep(1000);
                if (isRun) {
                    count--;
                    Message message = new Message();
                    message.what = 1;
                    message.arg1 = count;
                    handler.sendMessage(message);
                    if (count == 0) {
                        message = new Message();
                        message.what = 2;
                        message.obj = "Finish!!";
                        handler.sendMessage(message);                    loopFlag = false;
                    }
                }
            }
        } catch (Exception e) {
        }
    }
}
```

# 16.1 ANR과 스레드-핸들러

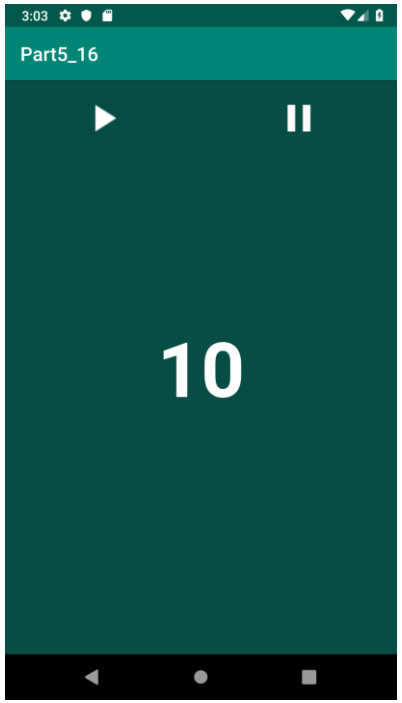
- sendMessage( ) 함수로 요청을 처리하려면 Handler 클래스의 서브 클래스를 정의

```
Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        if (msg.what == 1) {
            textView.setText(String.valueOf(msg.arg1));
        } else if (msg.what == 2) {
            textView.setText((String) msg.obj);
        }
    }
};
```

- sendMessage( ) 함수로 요청을 처리하려면 Handler 클래스의 서브 클래스를 정의

# Step by Step 16-1 – Thread - Handler

- 모듈 생성
- 파일 복사
- MainActivity 작성
- 실행





# 16.2 AsyncTask와 Looper

## 16.2.1. AyncTask

- 스레드-핸들러의 추상화 개념
- AyncTask를 상속받는 클래스를 작성

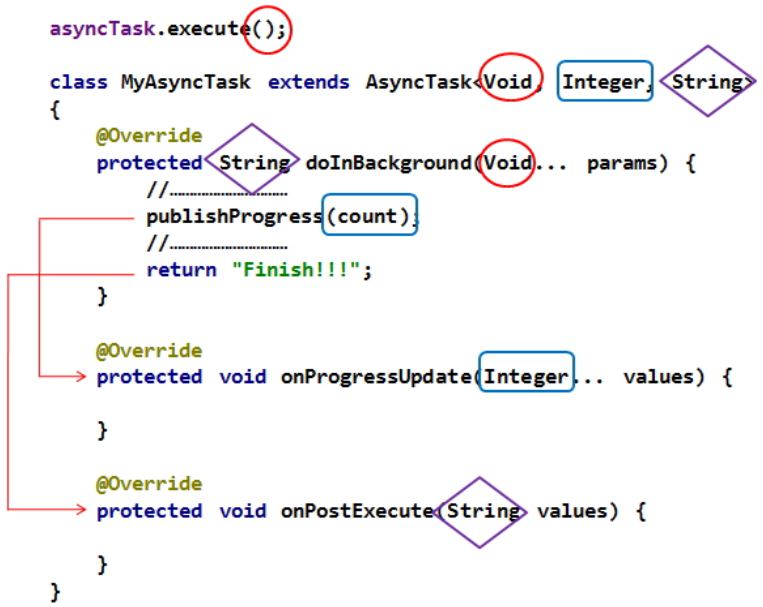
- doInBackground(Params... params): 스레드에 의해 처리될 내용
- onPreExecute(): AsyncTask의 작업을 시작하기 전에 호출.
- onPostExecute(Result result): AsyncTask의 모든 작업이 완료된 후 가장 마지막에 한 번 호출. doInBackground() 함 수의 최종 값을 받기 위해 사용
- onProgressUpdate(Progress... values): doInBackground() 함수에 의해 처리되는 중간중간 값을 받아 처리하기 위해 호출. doInBackground() 함수에서 publishProgress() 함수로 넘긴 값이 전달

```
class MyAsyncTask extends AsyncTask<Void, Integer, String> {

    @Override
    protected String doInBackground(Void... params) {
        //...    }

    @Override
    protected void onProgressUpdate(Integer... values) {
    }

    @Override
    protected void onPostExecute(String values) {
    }
}
```

# 16.2 AsyncTask와 Looper

class MyAsyncTask extends AsyncTask<Void, Integer, String>

- 첫 번째 타입: 백그라운드 작업을 위한 doInBackground() 함수의 매개변수 타입
- 두 번째 타입: doInBackground() 함수 수행에 의해 발생한 데이터를 publishProgress() 함수를 이용해 전달하는데 이때 전달 데이터 타입.
- 세 번째 타입: onPostExecute() 함수의 매개변수 타입과 동일하게 지정 doInBackground() 함수의 반환형

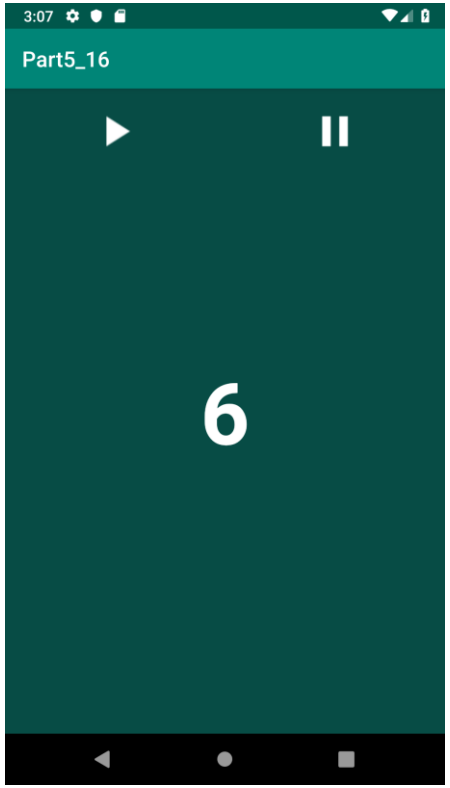



- AsyncTask 클래스를 수행하려면 클래스를 생성한 다음 execute( ) 함수를 호출

```
MyAsyncTask task=new MyAsyncTask();
task.execute();
```

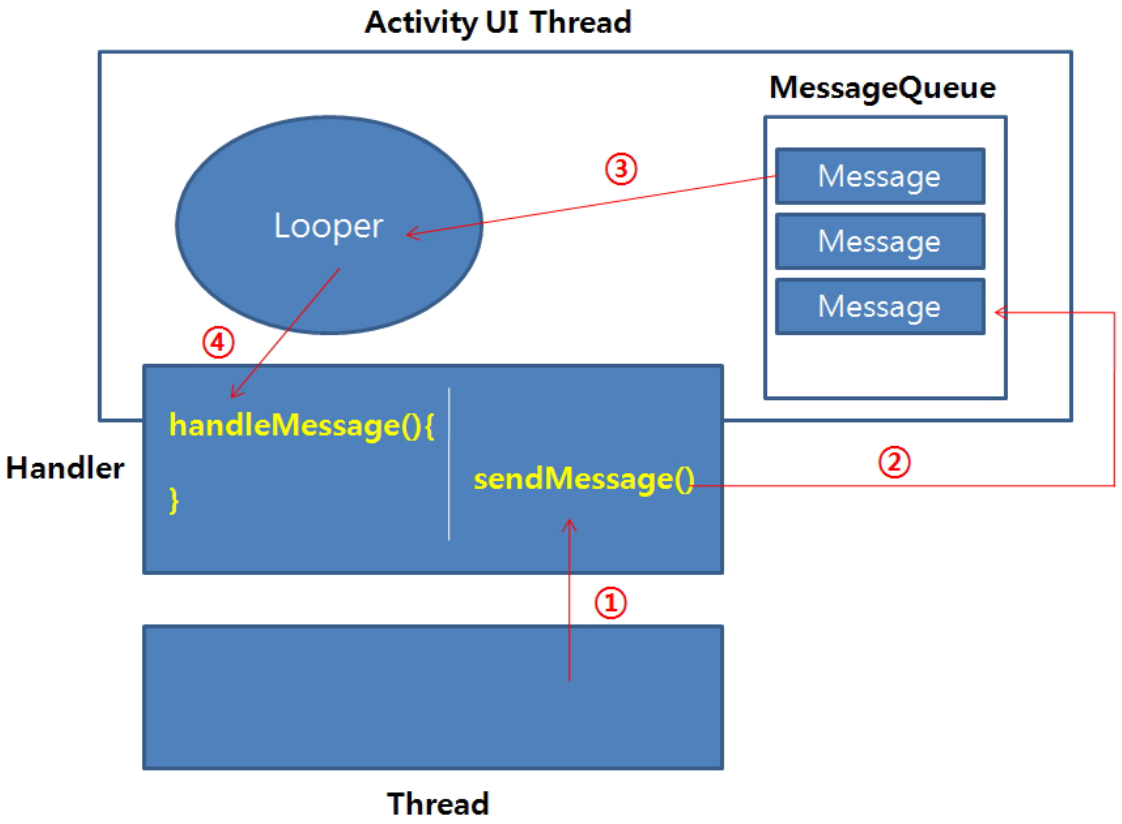
# Step by Step 16-2 - AsyncTask

- 액티비티 생성
- 파일 복사
- Lab16_2Activity 작성
- 실행



# 16.2 AsyncTask와 Looper

## 16.2.2. Looper




- 개발자 스레드 간의 연동을 핸들러 구조로 개발하려는 경우가 자바 코드로 직접 Looper를 준비

# 16.2 AsyncTask와 Looper

```
class OneThread extends Thread {
    Handler oneHandler;
    @Override
    public void run() {
        Looper.prepare();
        oneHandler = new Handler(){
            @Override
            public void handleMessage(Message msg) {
                int data=msg.arg1;
                if(msg.what==0){
                    Log.d("kkang", "even data : "+data);
                }else if(msg.what==1){
                    Log.d("kkang", "odd data : "+data);
                }
            }
        };
        Looper.loop();
    }
}
```

- Looper 종료

```
protected void onDestroy() {
    super.onDestroy();
    oneThread.oneHandler.getLooper().quit();
}
```

# 16.2 AsyncTask와 Looper

- Looper 이용

```
class TwoThread extends Thread {
    @Override
    public void run() {
        Random random=new Random();
        for(int i=0;i<10;i++){
            int data=random.nextInt(10);
            Message message=new Message();
            if(data % 2 == 0) {
                message.what = 0;
            }else {
                message.what=1;
            }
            message.arg1=data;
            oneThread.oneHandler.sendMessage(message);
        }
    }
}
```

# 16.2 AsyncTask와 Looper

- post ( ) 함수에 의한 요청을 처리하는 Looper를 내장한 HandlerThread 라이브러리 클래스를 이용 가능

```
handlerThread=new HandlerThread("HandlerThread");
handlerThread.start();
looperHandler=new Handler(handlerThread.getLooper());
```

- 핸들러 이용

```
looperHandler.post(new Runnable() {
    @Override
    public void run() {
        Log.d("kkang", "btn1...run..");
    }
});
```

# Step by Step 16-3 - Looper

- 액티비티 생성
- 파일 복사
- Lab16_3Activity 작성
- 실행