



자바응용SW(앱)개발자양성

## 액티비티 실행과 인텐트

백제직업전문학교

김영준 강사



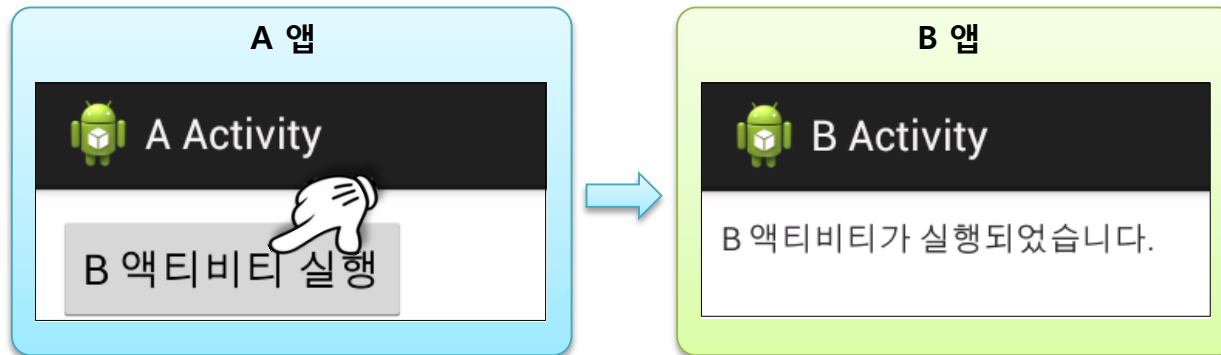
1.

액티비티 실행

## 액티비티 실행



■ 다음의 예제를 구현해본다.



## 액티비티 실행

### ■ 실행될 B 앱을 먼저 구현한다.

#### AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.superdroid.test.activity.b"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="16" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name="com.superdroid.test.activity.b.BActivity"
            android:label="@string/app_name" >

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>

    </application>

</manifest>
```

#### B 앱



#### B Activity

B 액티비티가 실행되었습니다.

#### 참고

```
<activity android:name="com.superdroid.test.activity.b.BActivity">
다음과 같이 패키지명을 생략하고 사용할 수 있다.
<activity android:name=".BActivity">
```

## 액티비티 실행

### ■ 실행될 B 앱을 먼저 구현한다.

#### res/layout/activity\_b\_layout.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="B 액티비티가 실행되었습니다." />

</LinearLayout>
```

#### src/BActivity.java

```
public class BActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_b_layout );
    }
}
```

#### B 앱



#### B Activity

B 액티비티가 실행되었습니다.

## 액티비티 실행

### ■ A 앱을 구현한다.

src/AActivity.java

```
public class AActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        ...
    }

    public void onClick( View v )
    {
        // ① 인텐트 하나를 생성한다.
        // =====
        Intent intent = new Intent();
        // =====

        // ② 인텐트에 실행할 패키지의 액티비티 정보를 설정한다.
        // =====
        ComponentName componentName = new ComponentName (
            "com.superdroid.test.activity.b",
            "com.superdroid.test.activity.b.BActivity" );

        intent.setComponent( componentName );
        // =====

        // ③ B 액티비티를 실행한다.
        // =====
        startActivity( intent );
        // =====
    } ...
}
```

A 앱



A Activity

B 액티비티 실행



## 액티비티 실행

### ■ A 앱을 구현한다.

#### res/layout/activity\_a\_layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="B 액티비티 실행"
        android:onClick="onClick"/>

</LinearLayout>
```

#### src/AActivity.java

```
public class AActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        ...

    public void onClick( View v )
    {
        // ① 인텐트 하나를 생성한다.
        // =====
        Intent intent = new Intent();
        // =====

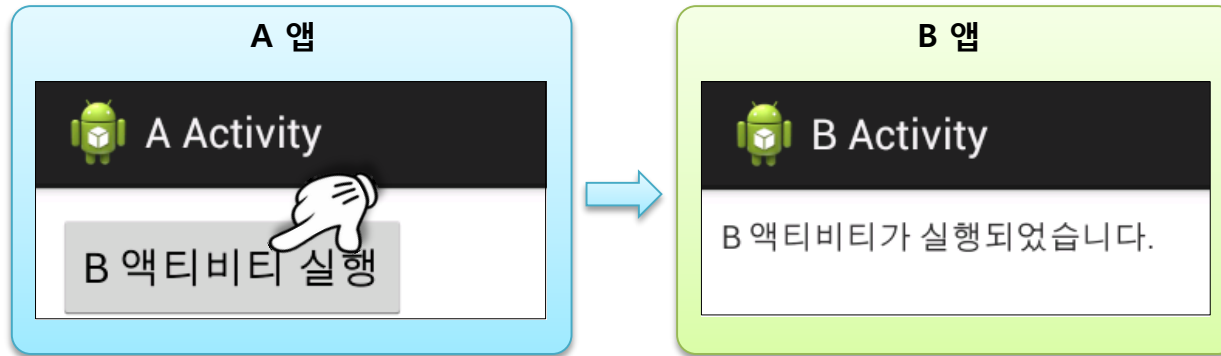
        // ② 인텐트에 실행할 패키지의 액티비티 정보를 설정한다.
        // =====
        ComponentName componentName = new ComponentName (
            "com.superdroid.test.activity.b",
            "com.superdroid.test.activity.b.BActivity" );

        intent.setComponent( componentName );
        // =====

        // ③ B 액티비티를 실행한다.
        // =====
        startActivity( intent );
        // =====
    } ...
}
```

## 액티비티 실행

### ■ A 앱을 실행



별 어려움 없이 startActivity 함수와 실행될 액티비티 정보가 담긴 인텐트만으로 원하는 액티비티를 실행할 수 있었다.

어떻게 A 액티비티가 B 패키지명과 액티비티명만으로 B 액티비티를 실행할 수 있을까?

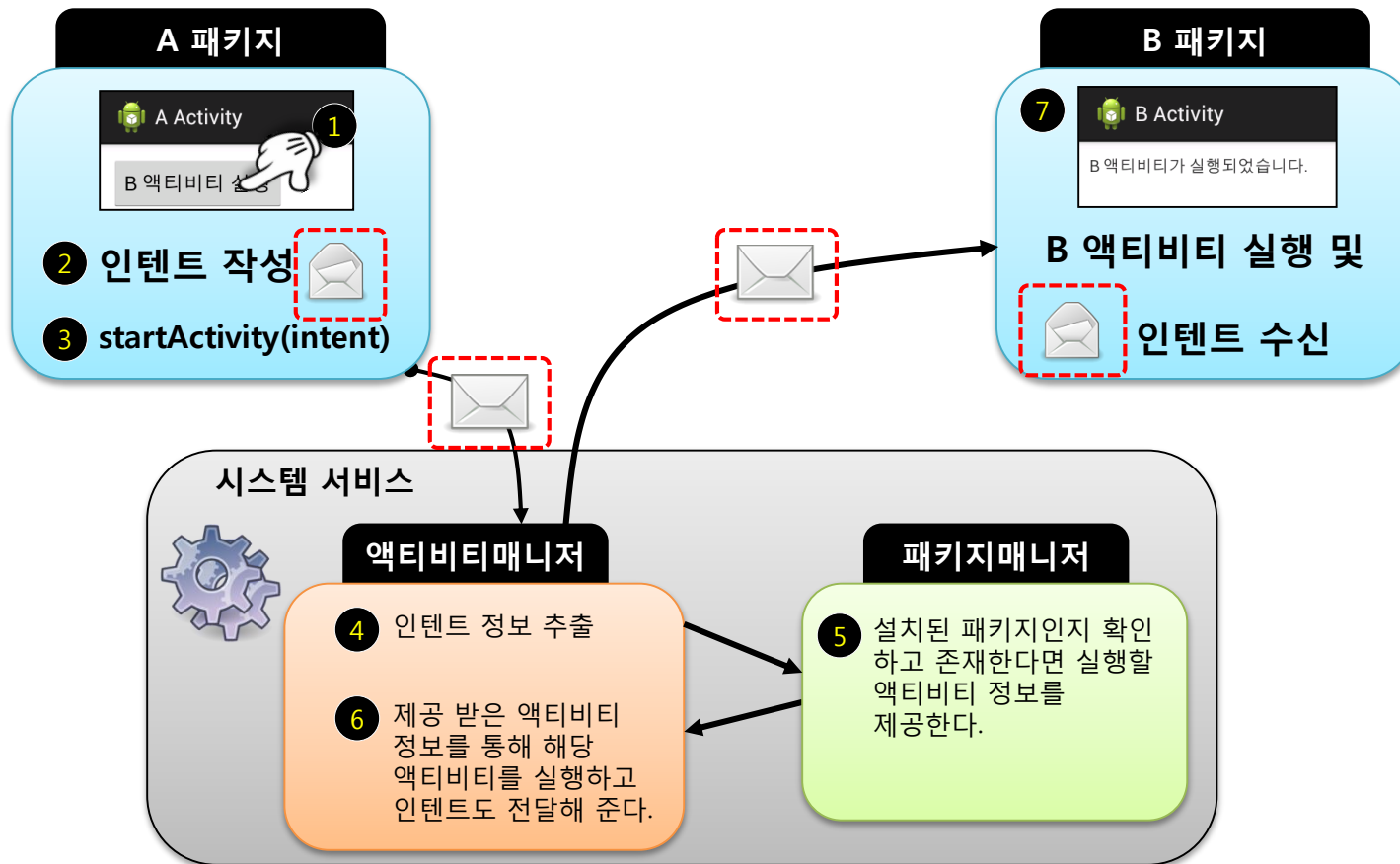
그건 바로 시스템 서비스인 **패키지 매니저** Package Manager와 **액티비티 매니저** Activity Manager가 있었기 때문이다.

다음은 두 가지 시스템 서비스의 역할에 대해서 알아보자.



## 인텐트 그리고 패키지 매니저와 액티비티 매니저

- **패키지 매니저** : 앱을 설치하고 삭제하며, 설치된 모든 패키지 정보를 수집한다.  
수집된 정보는 원하는 앱에게 제공할 수 있다. 모든 패키지 정보는 각 패키지마다 존재하는 AndroidManifest.xml 파일의 내용을 기반으로 설치 시점에 수집되었다.
- **액티비티 매니저** : 안드로이드의 4대 컴포넌트들을 관리하며,  
그중 액티비티를 실행하는 기능도 가진다.



## 인텐트 그리고 패키지 매니저와 액티비티 매니저

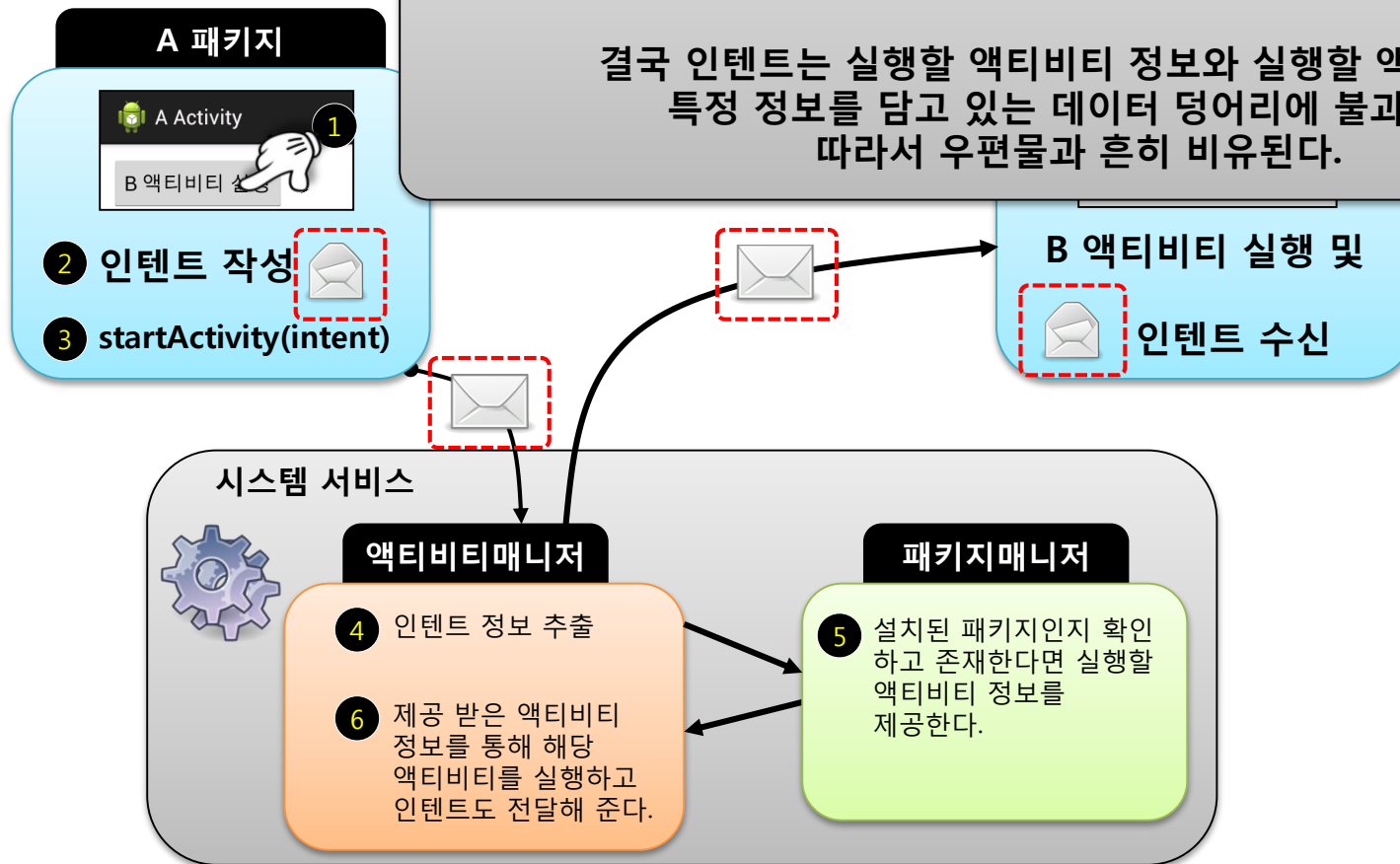
- 패키지 매니저 : 앱을 설치할 수 있는 권한을 가진 패키지
- 액티비티 매니저 : 안드로이드 시스템에서 실행 중인 액티비티를 관리



인텐트는 액티비티를 실행하고 역할이 끝났는데 왜 B 액티비티에 전달하는거지?

- 사실 액티비티 실행은 액티비티 매니저가 처리한다. 다만 인텐트는 실행할 액티비티 정보만 담고 있다.
- 목적지 액티비티에 인텐트를 전달하는 것은 실행한 액티비티가 목적지 액티비티에 특정 데이터를 전달하기 위함이다.

결국 인텐트는 실행할 액티비티 정보와 실행할 액티비티에 특정 정보를 담고 있는 데이터 덩어리에 불과하다. 따라서 우편물과 흔히 비유된다.



## 실행될 액티비티에게 데이터 전달하기

### ■ A 액티비티가 B액티비티로 전달할 데이터를 담아서 보내기

src/AActivity.java

```
public void onClick( View v )
{
    ...
    intent.putExtra( "NAME", "Superdroid" );
    startActivity( intent );
}
```

### ■ B 액티비티가 A액티비티에게 전달받은 데이터 확인하기

src/BActivity.java

```
public class BActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        ...
        Intent intent = getIntent();
        String name = intent.getStringExtra( "NAME" );

        TextView receivedStr = (TextView)findViewById(
                                                    R.id.intent_received_data
        );
        receivedStr.setText( "" + name );
    }
}
```

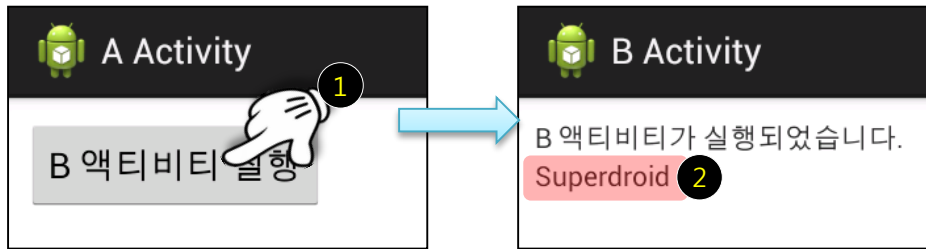


B Activity

B 액티비티가 실행되었습니다.

Superdroid

## 실행될 액티비티에게 데이터 전달하기



■ 이쯤에서 한 가지 의문이 생긴다. 서로 다른 프로세스는 서로의 메모리를 절대 참조할 수 없다.

DDMS - BActivity/res/values/strings.xml - Eclipse SDK

File Edit Navigate Search Project Run Window Help

Devices

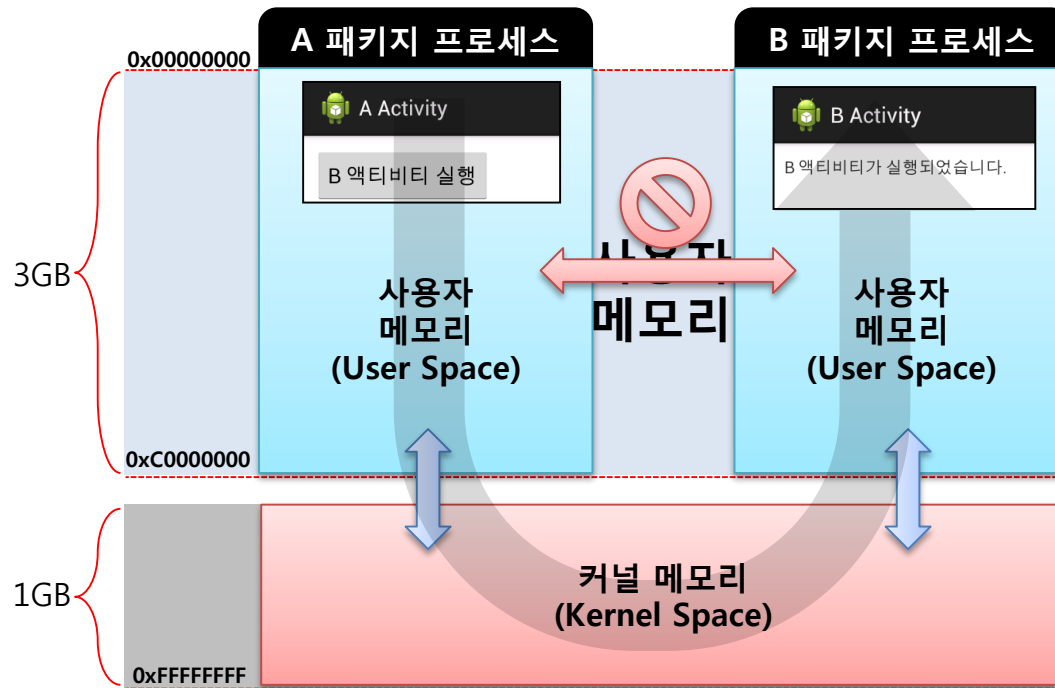
Name	PID	PPID
MyDevice_API16_WVGA [emulator-5] Online		MyDevice_...
android.process.acore	305	8600
com.android.providers.calendar	411	8601
com.android.systemui	204	8602
system_process	149	8616
com.superdroid.test.activity.a	735	8606
com.superdroid.test.activity.b	989	8608

안드로이드에서는 하나의 패키지가 실행되면 하나의 프로세스가 생성되며, 패키지의 모든 처리는 생성된 프로세스에서 돌아간다.

예제에서 두 가지 패키지가 실행되었으므로 두 개의 프로세스가 돌아가고 있는 것이다.

## 프로세스 간 데이터 전달

- 서로 다른 프로세스는 데이터를 서로 공유할 수 없다. 그런데 어떻게 인텐트를 목적지 액티비티에게 전달했을까?



안드로이드는 32 비트 리눅스 커널을 사용하므로  
4GB 이내의 메모리 주소 공간을 부여할 수 있다.

다른 프로세스가 메모리를 침범하여  
자신의 프로세스에 영향을 주는 것을  
막기 위함이다.

이러한 구조로 인해 특정 프로세스가  
오동작으로 죽어도 다른 프로세스는  
영향을 받지 않는다.

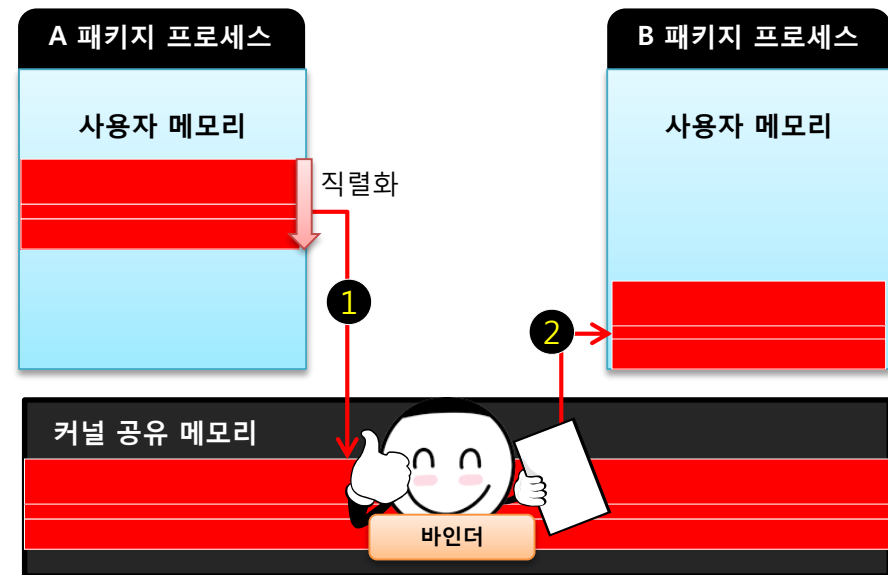
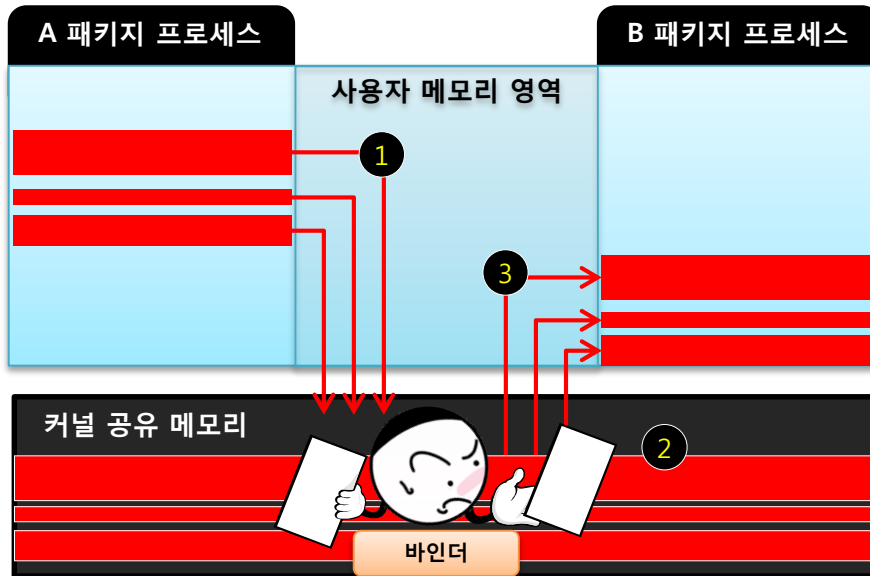
참고로 프로세스 간 데이터를 주고받은 것을 전  
문 용어로 IPC(Inter Process Communication)  
라고 한다. IPC는 안드로이드 이외에 다른 멀티  
프로세스 운영체제에서도 방법의 차이만 있을 뿐  
모두 존재한다.

## 프로세스 간 데이터 전달

### ■ 안드로이드의 프로세스 간 데이터 전달

같이 세 가지 프리미티브<sup>Primitive</sup> 타입(int, float, double...)의 변수를 전달한다고 가정한다.

- ① String strData = "안녕하세요"
- ② int intData = 100
- ③ long longData = 100L;



## 프로세스 간 데이터 전달

- IPC는 메모리에 직렬화가 기반이되어야 한다. 여기서 인텐트 역시도 직렬화가 적용된 객체다. 따라서 인텐트를 제대로 이해하려면 직렬화에 대해 이해해야 한다.

직렬화 객체 인텐트는 다음과 같이 패키지로 묶어 이해해야 할 객체들이 있다.



IPC는 안드로이드 전반에 걸쳐 사용되고 위의 객체들은 매우 빈번히 활용된다.

2.

직렬화 객체 인텐트



## 데이터 덩어리 인텐트

안드로이드에서 사용하는 인텐트는 사실 **데이터 덩어리**에 불과하다. 즉 인텐트 클래스 내부적으로 특별한 기능이 존재하지 않다는 의미다. 다만 특징이라면 인텐트에 수많은 프리미티브 타입 데이터와 그 밖에 직렬화된 데이터를 담아 하나로 묶어 직렬화 데이터 덩어리로 만들 수 있다는 점이다.

따라서 여러분들도 이 절에서는 인텐트를 단순히 **직렬화된 데이터 덩어리**로만 봐주길 바란다.

다음의 순서대로 직렬화 객체를 살펴보자.



## 프리티비브 타입 직렬화 객체



프리티비브 타입이란 boolean, byte, char, short, int, long, float, String 등과 같이 가공되지 않은 순수 자료형을 말한다.

이러한 자료형은 각각 메모리에 분리되어 할당되지 않는다. 예를 들면 int 형은 4바이트를 사용하는 자료형이다. 만일 int a = 0 이라고 변수를 선언하면 메모리에는 4바이트 연속된 메모리에 할당해주는 것이다. 그러므로 프리미티브 타입의 객체는 그 자체가 직렬화 객체며, 별도의 처리를 하지 않고 다른 프로세스에 전달이 가능한 상태다.

프리티비브 타입의 변수 자체를 전달하는 경우는 드물고 대부분 수많은 프리미티브 타입의 변수를 클래스 내에 정의하고 객체화한다.

```
class SampleClass
{
    public int      mIntData = 0;
    public String   mStrData = "Superdroid";
    public long     mLongData = 100L;
}
```

하지만 문제는 클래스를 객체화하면 내부의 프리미티브 타입 변수들이 서로 연속된 메모리에 할당되지 않아 직렬화 객체가 될 수 없다는 점이다.

따라서 SampleClass 클래스를 직렬화할 방법이 필요하다. 이를 위해 **자바의 Serializable 인터페이스를 사용해야 한다.**

## 자바의 Serializable 인터페이스를 상속받은 직렬화 객체



■ 우선 직렬화되지 않은 클래스를 살펴보자.

```
public class SampleData
{
    private int    mData = 0;
    private String mData = "Superdroid";

    public int getIntData()
    {
        return mData;
    }

    public String getStringData()
    {
        return mData;
    }

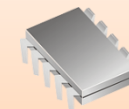
    public void setIntData( int intData )
    {
        mData = intData;
    }

    public void setStringData( String strData )
    {
        mData = strData;
    }
}
```

소스 코드

클래스 파일

값이 변하는  
멤버 변수



mIntData

0

"Superdroid"

mStrData

직렬화 메모리  
할당

! 너무 당연한 얘기

SimpleData 객체를 다른 프로세스로 전달할 때  
클래스 파일을 다른 프로세스 앱 개발자에게 주고  
변수들만 IPC를 통해 보낸다.

너무 당연한 얘기지만 이것이 바로 직렬화 객체 전달의  
기본 원리다.

## 자바의 Serializable 인터페이스를 상속받은 직렬화 객체

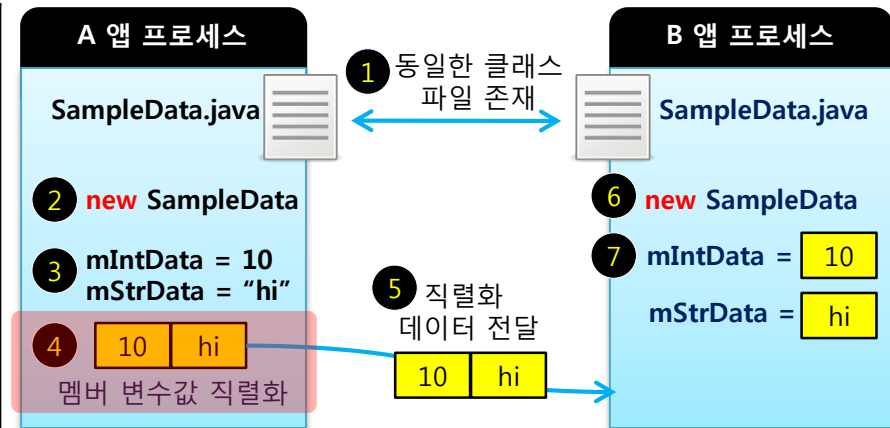
```
public class SampleData
{
    private int    mIntData = 0;
    private String mStrData = "Superdroid";

    public int getIntData()
    {
        return mIntData;
    }

    public String getStringData()
    {
        return mStrData;
    }

    public void setIntData( int intData )
    {
        mIntData = intData;
    }

    public void setStringData( String strData )
    {
        mStrData = strData;
    }
}
```



멤버 변수들을 직렬화 하는 방법만 알면 되겠다.

## 자바의 Serializable 인터페이스를 상속받은 직렬화 객체

■ 클래스의 멤버 변수를 직렬화하는 것은 매우 간단하다. 그냥 Serializable 객체만 상속 받으면 된다.

```
public class SampleData implements Serializable
```

```
{
```

```
    private static final long serialVersionUID = 1000000L;
```

```
    private int    mData = 0;
```

```
    private String mData = "Superdroid";
```

```
    public int getIntData()
```

```
    {
        return mData;
    }
```

```
    public String getStringData()
```

```
    {
        return mData;
    }
```

```
    public void setIntData( int intData )
```

```
    {
        mData = intData;
    }
```

```
    public void setStringData( String strData )
```

```
    {
        mData = strData;
    }
}
```

Serializable이 뭐길래?



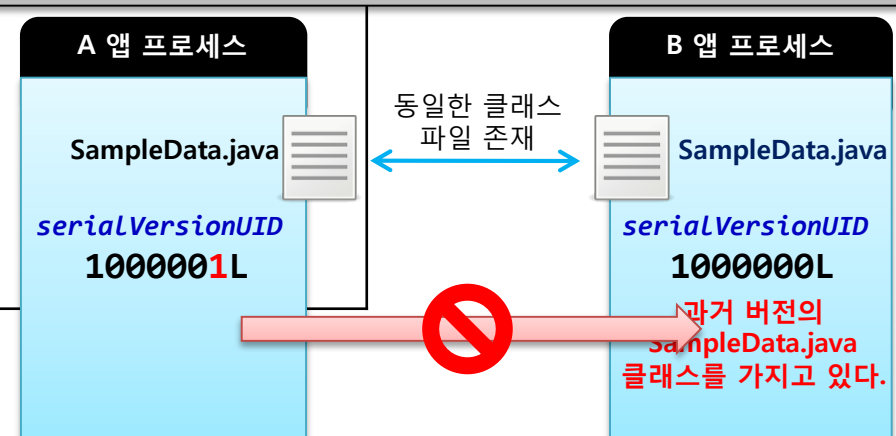
자바의 Serializable.java

```
public interface Serializable {
}
```

보다시피 인터페이스만 정의되어 있을 뿐 아무런 내용이 없다. 이런 클래스를 **마커인터페이스** **MarkerInterface**라고 한다.

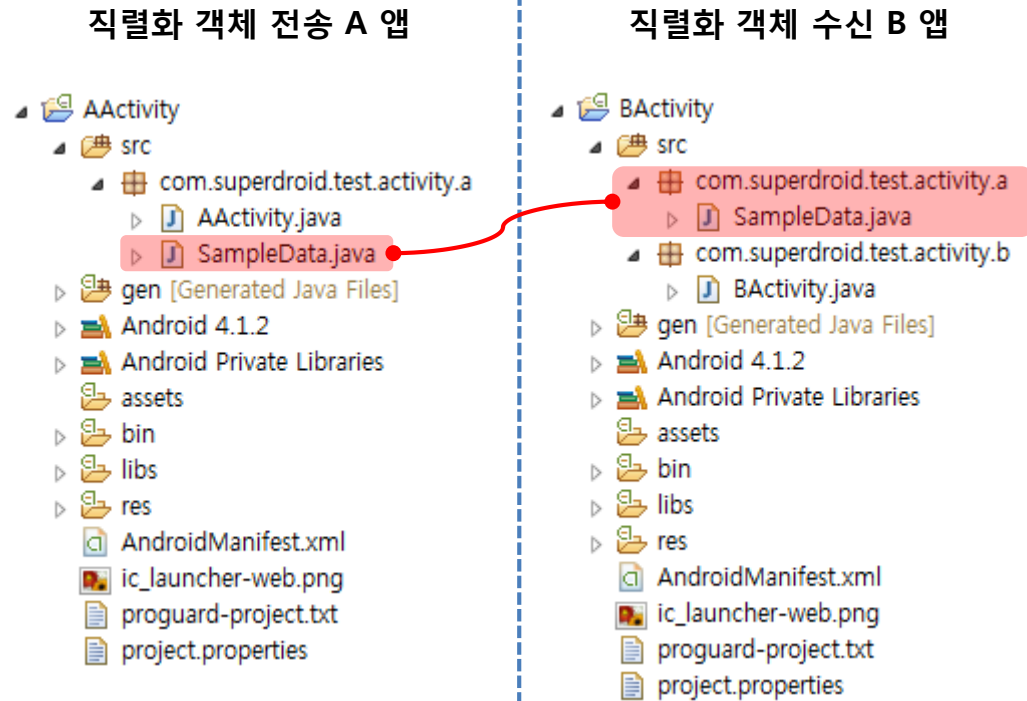
즉 클래스를 객체화할 때 내부 변수들을 직렬화해야 한다고 알리는 용도이다.

시스템은 클래스가 Serializable 인터페이스를 상속받았다면 내부의 멤버변수들을 모두 직렬화하여 객체를 만든다. 그러므로 개발자는 Serializable 인터페이스를 상속받아 직렬화 객체임을 알리면 그만이다.



## 자바의 Serializable 인터페이스를 상속받은 직렬화 객체

### ■ Serializable 객체를 직접 전송해보자.



## 자바의 Serializable 인터페이스를 상속받은 직렬화 객체

### src/AActivity.java

```
public class AActivity extends Activity
{
    ...

    public void onClick( View v )
    {
        // ① SampleData 객체를 생성하고 멤버 변수를 변경한다.
        SampleData sampleData = new SampleData();
        sampleData.setIntData(123456789);
        sampleData.setStringData("Serializable Object");

        // ② SampleData 객체를 인텐트에 추가한다.
        intent.putExtra( "SAMPLE_DATA", sampleData );

        // ③ B 앱 액티비티에 인텐트를 전달과 동시에 실행한다.
        startActivity( intent );
    }
}
```

## 자바의 Serializable 인터페이스를 상속받은 직렬화 객체

src/BActivity.java

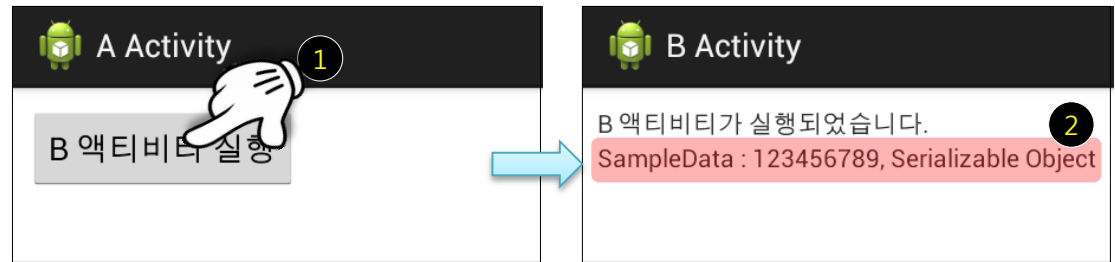
```
public class BActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_b_layout );

        // ① 자신을 호출한 액티비티가 보낸 인텐트에 직렬화 객체를 추출한다.
        Intent intent = getIntent();
        SampleData sampleData =
            (SampleData)intent.getSerializableExtra("SAMPLE_DATA" );

        if( sampleData == null )
        {
            Toast.makeText( this,
                "SampleData Null!",
                Toast.LENGTH_LONG ).show();

            return;
        }

        // ② 전달 받은 직렬화 객체 내용을 출력한다.
        TextView receivedStr = (TextView)findViewById(
            R.id.intent_received_data );
        receivedStr.setText( "SampleData : " + sampleData.getIntData() +
            ", " + sampleData.getStringData() );
    }
}
```

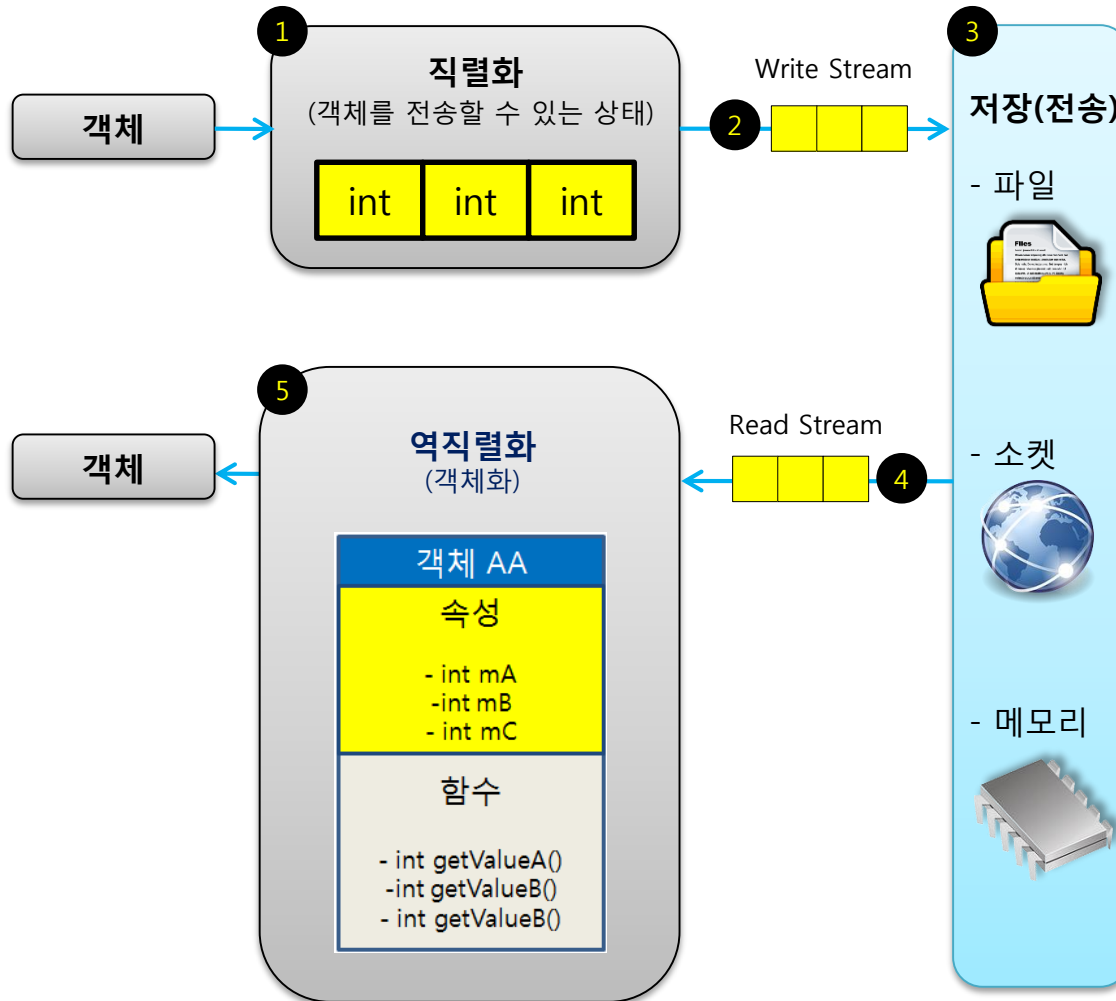




# 자바의 Serializable 인터페이스를 상속받은 직렬화 객체

## ■ Serializable 활용

Serializable 인터페이스는 프로세스 간 통신을 위해서만 활용되는 것은 아니다. 예를 들면 파일에 쓰거나 네트워크를 이용해 다른 단말기나 서버로 전송할 때 활용되기도 한다. 이 과정을 그림으로 표현해보자.





## ■ Parcel 클래스는 안드로이드에서 IPC 전용 데이터로 사용하기 위해 만들어진 클래스다.

따라서 프로세스 간 데이터 전달에 최적화되어 속도가 매우 빠르다. 그렇다면 무조건 Parcel 클래스를 사용해야 할까? 안드로이드는 굳이 Parcel 객체를 사용하지 않아도 내부적으로 전송되는 데이터들은 Parcel 객체로 변환하여 전송한다. 다만 직접 Parcel 객체를 사용하면 변환 과정을 거치지 않으므로 좀 더 성능이 좋다.

### Parcel 클래스가 제공하는 몇 가지 함수

#### Parcel 객체를 반환하는 함수

- **public static** Parcel obtain()

#### 각종 직렬화 객체를 쓰는 함수들

- **public final void** writeInt(**int** val)
- **public final void** writeLong(**long** val)
- **public final void** writeString(String val)
- **public final void** writeSerializable(Serializable s)
- ...

#### 각종 직렬화 객체를 읽는 함수들

- **public final int** readInt()
- **public final long** readLong()
- **public final** String readString()
- **public final** Serializable readSerializable()
- ...

# Parcel

## ■ 간단한 예제를 통해 이해해보자.

src/BActivity.java

```
public class BActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_b_layout );

        // ① Parcel 객체를 생성하고 프리미티브 타입의 데이터를 쓴다.
        // =====
        Parcel parcel = Parcel.obtain();
        parcel.writeInt(10);
        parcel.writeString("Superdroid");
        // =====

        // ② Parcel 객체에 저장된 프리미티브 타입의 데이터를 읽어서 화면에 출력한다.
        // =====
        TextView receivedStr = (TextView)findViewById(
                                                    R.id.intent_received_data );

        parcel.setDataPosition( 0 );
        receivedStr.setText( "SampleData : " + parcel.readInt() + ", " +
                            parcel.readString() );
        // =====
    }
}
```



B Activity

B 액티비티가 실행되었습니다.  
SampleData : 10, Superdroid

# Parcel


## Parcel의 주의 사항

1 Parcel 객체에 데이터를 쓴 순서

- 1) `parcel.writeInt( 10 );`
- 2) `parcel.writeString( "Superdroid" );`

2 쓴 순서대로 읽음


- 1) `parcel.readInt( );`
- 2) `parcel.readString( )`

 B Activity

B 액티비티가 실행되었습니다.  
SampleData : 10, Superdroid

3 쓴 순서의 반대로 읽음

- 1) `parcel.readString( )`
- 2) `parcel.readInt( );`

 B Activity

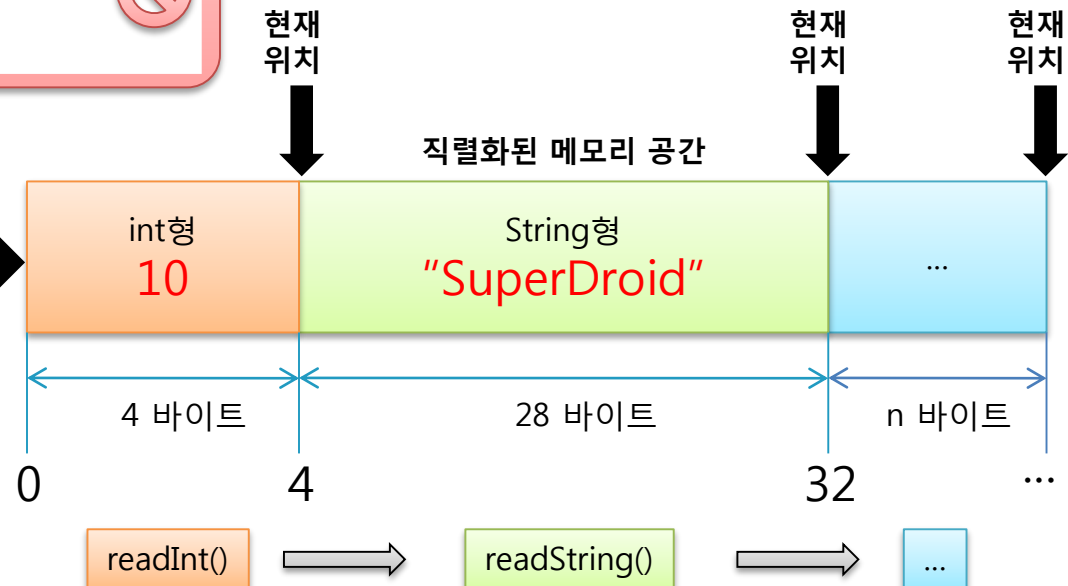
B 액티비티가 실행되었습니다.  
SampleData :  
Superdro, 0

왜 이상한 결과가 나왔을까?

잠시 Parcel의 구조를 살펴보자.

Parcel 객체를 사용함에 있어 순서를 유지해야 한다는 것은 너무 불편하다.

따라서 Parcel은 단독으로 사용되지 않고 Parcelable을 통한다.



# Parcelable



- Parcelable 클래스는 일종의 Parcel 데이터를 사용하는 설명서와도 같다.  
그 이유는 Parcelable 클래스 내부에 Parcel 데이터를 읽고 쓰는 함수들을 구현하기 때문이다.
- Parcelable 클래스를 구현해보자.

# Parcelable

src/SampleData.java

```
public class SampleData implements Parcelable
{
    private int      mIntData = 0;
    private String   mStrData = "Superdroid";

    public int getIntData()
    {
        return mIntData;
    }

    public String getStringData()
    {
        return mStrData;
    }

    public void setIntData( int intData )
    {
        mIntData = intData;
    }

    public void setStringData( String strData )
    {
        mStrData = strData;
    }

    @Override
    public int describeContents()
    {
        return 0;
    }
}
```

# Parcelable

```
// 송신 측 사용함수
@Override
public void writeToParcel( Parcel dest, int flags )
{
    dest.writeInt( mIntData );
    dest.writeString( mStrData );
}

public static final Parcelable.Creator<SampleData> CREATOR = new Creator<SampleData>()
{
    // 수신 측 사용 함수
    @Override
    public SampleData createFromParcel( Parcel src )
    {
        sampleData sampleData = new SampleData();

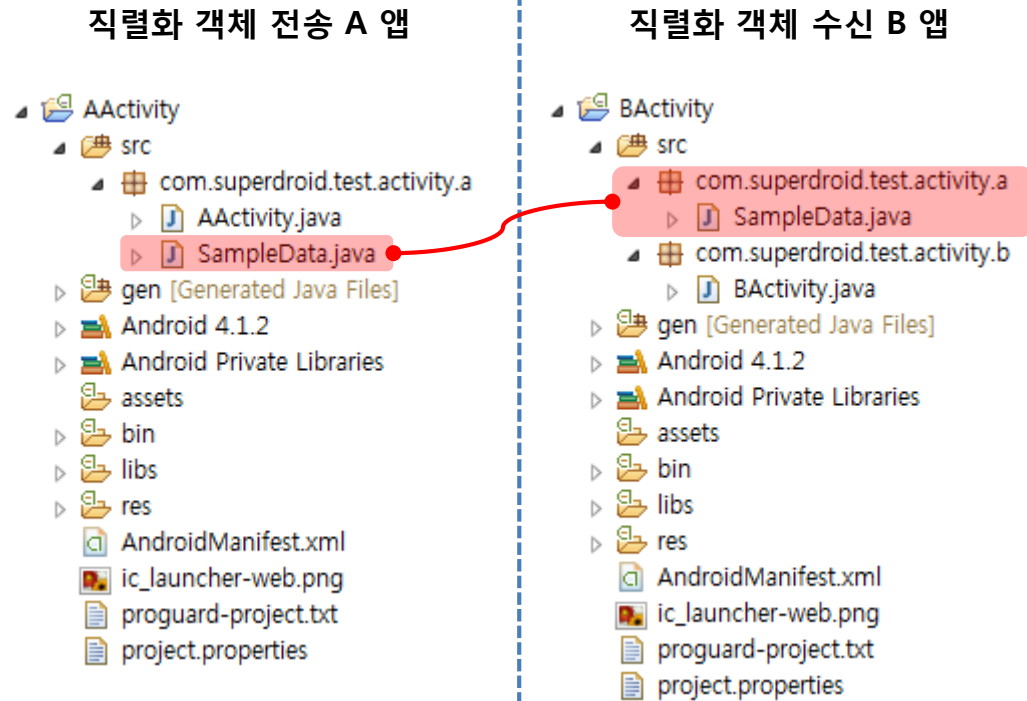
        sampleData.setIntData( src.readInt() );
        sampleData.setStringData( src.readString() );

        return sampleData;
    }

    @Override
    public SampleData[] newArray( int size )
    {
        return null;
    }
};
}
```

# Parcelable

■ Parcelable 클래스도 Serializable과 같이 클래스 파일을 수신측에 포함시켜야 한다.





# Parcelable

## src/AActivity.java

```
public class AActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_a_layout );
    }

    public void onClick( View v )
    {
        ...

        // Parcelable 객체를 생성 및 데이터를 변경하고 실행될 액티비티에게 전달할
        // 인텐트에 추가한다.
        // =====
        SampleData sampleData = new SampleData();
        sampleData.setIntData(123456789);
        sampleData.setStringData("Parcelable Object");

        intent.putExtra( "SAMPLE_DATA", sampleData );
        // =====

        startActivity( intent );
    }
}
```

# Parcelable

src/AActivity.java

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

        public void onClick(View v) {
            ...

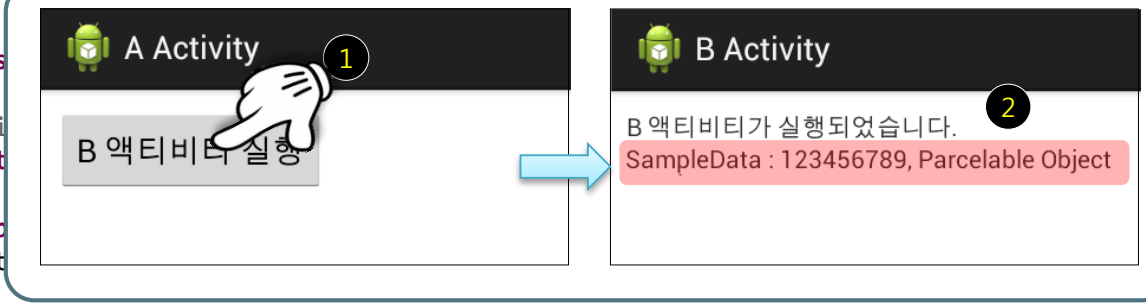
            // Parcelable 객체
            // 인텐트에 추가한다
            // =====
            SampleData sampleData = new SampleData();
            sampleData.setIntData(123456789);
            sampleData.setStringData("Parcelable Object");

            intent.putExtra("SAMPLE_DATA", sampleData);
            // =====

            startActivity(intent);
        }
    }
}
```

src/BActivity.java

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



```
// ① 자신을 호출한 액티비티가 보낸 인텐트에 직렬화 객체를 추출한다.
// =====
Intent intent = getIntent();
SampleData sampleData =
    (SampleData)intent.getParcelableExtra( "SAMPLE_DATA" );

if( sampleData == null )
{
    Toast.makeText( this,
        "SampleData Null!",
        Toast.LENGTH_LONG ).show();

    return;
}
// =====
// ② 전달 받은 직렬화 객체 내용을 출력한다.
// =====
TextView receivedStr = (TextView)findViewById(
    R.id.intent_received_data );
receivedStr.setText( "SampleData : " + sampleData.getIntData() +
    ", " + sampleData.getStringData() );
// =====
}
```

## Parcelable

■ **Parcelable**은 프리미티브 타입 변수 몇 개만 전달하는 경우 너무 불편하다.

1. Parcelable 클래스를 구현해야 한다.
2. 구현된 Parcelable 클래스 파일을 송신 측 앱에 배포해야 한다.
3. 만일 구현된 Parcelable 클래스가 변경되었다면 다시 송신 측 앱에 수정 Parcelable 클래스를 공유해야 한다.

이를 위해 안드로이드는 **Bundle** 클래스를 제공한다.

## 참 편리한 Bundle 클래스



```
public final class
Bundle
extends Object
implements Parcelable Cloneable
```

- 번들 클래스 Parcelable 클래스를 상속 받아 구현된 클래스다.
- 안드로이드 SDK 라이브러리에 포함되어 있기 때문에 클래스 파일 자체를 배포할 필요가 없다.
- 번들은 모든 타입의 직렬화 객체를 담을 수 있을 뿐만 아니라 Parcel과는 달리 읽고 쓰는 순서를 일치시킬 필요가 없다.

### Parcel 클래스가 제공하는 몇 가지 함수

각종 직렬화 객체를 쓰는 함수들

- **public void** putInt(**String** key, **int** val)
- **public void** putLong(**String** key, **long** val)
- **public void** putString(**String** key, String val)
- **public void** putSerializable(**String** key, Serializable s)
- ...

각종 직렬화 객체를 읽는 함수들

- **public int** getInt(**String** key)
- **public long** getLong(**String** key)
- **public String** getString(**String** key)
- **public Serializable** getSerializable(**String** key)
- ...

put, get 뒤에 타입명만 변경해서 쓰면 된다.

- 번들은 자바 컬렉션 프레임워크 JAVA Collection Framework의 Map과 같은 방식으로 데이터를 읽고 쓴다. 즉 키와 값 형식으로 사용한다. 실제로 번들 내부에는 맵을 사용한다.

■ Bundle 클래스를 사용해보자.

## 참 편리한 Bundle 클래스

src/AActivity.java

```
public class AActivity {
    @Override
    protected void onCreate() {
        super.onCreate();
        setContentView(R.layout.activity_main);
    }

    public void onClick() {
        ...

        Bundle bundleData = getIntent().getBundleExtra("SAMPLE_DATA");

        startActivity();
    }
}
```

src/BActivity.java

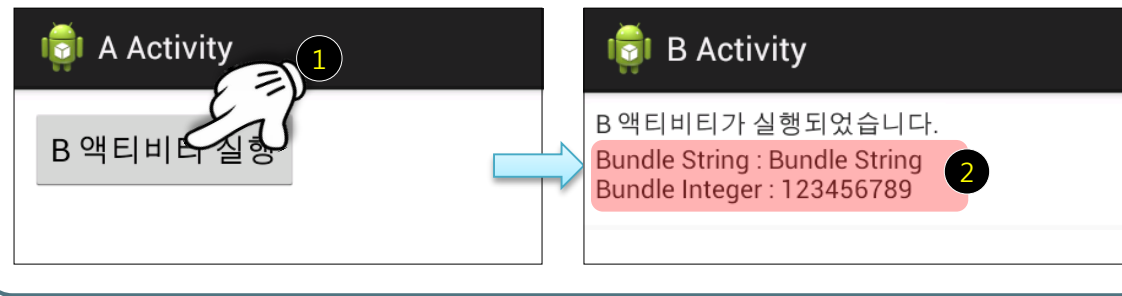
```
public class BActivity {
    @Override
    protected void onCreate() {
        super.onCreate();
        setContentView(R.layout.activity_main);

        Intent intent = getIntent();
        Bundle bundleData = intent.getBundleExtra("SAMPLE_DATA");

        if( bundleData == null )
        {
            Toast.makeText( this,
                           "Bundle Data Null!",
                           Toast.LENGTH_LONG ).show();

            return;
        }

        TextView receivedStr = (TextView)findViewById(
            R.id.intent_received_data );
        receivedStr.setText( "Bundle String : " +
                           bundleData.getString("STR_DATA") + "\n" +
                           "Bundle Integer : " +
                           bundleData.getInt("INT_DATA") );
    }
}
```



```
Intent intent = getIntent();
```

```
Bundle bundleData = intent.getBundleExtra( "SAMPLE_DATA" );
```

```
if( bundleData == null )
```

```
{
    Toast.makeText( this,
                   "Bundle Data Null!",
                   Toast.LENGTH_LONG ).show();

    return;
}
```

```
return;
```

```
TextView receivedStr = (TextView)findViewById(
    R.id.intent_received_data );
```

```
receivedStr.setText( "Bundle String : " +
                   bundleData.getString("STR_DATA") + "\n" +
                   "Bundle Integer : " +
                   bundleData.getInt("INT_DATA") );
```

## 참 편리한 Bundle 클래스

### src/BActivity.java

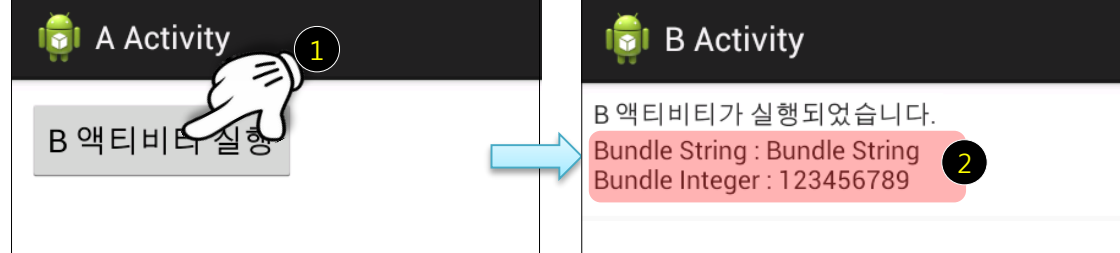
```
public class BActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_b_layout );

        Intent intent = getIntent();
        Bundle bundleData = intent.getBundleExtra( "SAMPLE_DATA" );

        if( bundleData == null )
        {
            Toast.makeText( this,
                           "Bundle Data Null!",
                           Toast.LENGTH_LONG ).show();

            return;
        }

        TextView receivedStr = (TextView)findViewById(
                                                    R.id.intent_received_data );
        receivedStr.setText( "Bundle String : " +
                           bundleData.getString( "STR_DATA" ) + "\n" +
                           "Bundle Integer : " +
                           bundleData.getInt( "INT_DATA" ) );
    }
}
```



## 데이터 덩어리에서 인텐트로 변신



```
public class
Intent
extends Object
implements Parcelable, Cloneable
```

- 인텐트도 Parcelable 클래스를 상속 받아 구현된 클래스다.  
참고로 안드로이드는 프로세스간 통신이 필요한 클래스는 대부분 Parcelable 클래스를 상속받아 구현한다.

### ■ 인텐트 내부에서 사용되는 멤버 변수를 살펴보자.

- **private** String mAction
- **private** Uri mData
- **private** String mType
- **private** String mPackage
- **private** ComponentName mComponent
- **private** int mFlags
- **private** HashSet<String> mCategories
- **private** Bundle mExtras

- 인텐트도 번들과 같이 각종 직렬화 객체를 포함시킬 수 있다. 바로 내부에 mExtras 번들 객체가 있기 때문이다.

각 변수들은 인텐트 클래스 측면에서 특별한 기능을 위해 존재하는 것이 아니며, 단지 직렬화된 데이터 덩어리다.

하지만 앱과 액티비티 매니저, 패키지 매니저 등의 시스템 서비스 간에 약속된 의미 있는 정보를 채워 넣으면서 인텐트 라는 것이 된다.

## 데이터 덩어리에서 인텐트로 변신

### ■ 인텐트에 의미를 부여하는 여섯 가지 정보

정보 분류	멤버변수	설명
컴포넌트 정보	String mPackage ComponentName mComponent	명시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
액션	String mAction	암시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
카테고리	HashSet<String> mCategories	
데이터 위치와 타입	Uri mData String mType	
엑스트라	Bundle mExtras	각종 컴포넌트에 순수 데이터를 전달하기 위한 용도
플래그	int mFlags	각종 컴포넌트를 제어하기 위한 플래그

인텐트 정보의 주 목적은 특정 컴포넌트를 실행 정보와 원하는 데이터를 전달하는 것이다.

이를 위해 인텐트의 여섯 가지 분류 정보가 어떻게 활용되는지 다음 강좌에서 설명한다.



## 직렬화 객체들의 정리



### Primitive 타입

프리미티브 타입은 순수 자료형이기 때문에 클래스의 멤버 변수로 존재한다. 따라서 직접 프로세스 통신을 위한 객체로 활용되지 않는다.

### Serializable

클래스 자체를 직렬화 객체로 만들 수 있다. 하지만 클래스 파일을 송신 측에 배포해야 하고 안드로이드 전용 직렬화 객체가 아니기 때문에 다소 느리다.

### Parcel

안드로이드 IPC를 담당하는 바인더가 사용하는 직렬화 객체다. 따라서 전송 속도가 매우 빠르다. 하지만 데이터를 읽고 쓰는 순서를 지켜야 한다. 단독으로 사용되지 않는다.

### Parcelable

Parcel 객체의 데이터를 읽고 쓰는 순서를 정의한 클래스다. Serializable과 같이 클래스 파일을 송신 측에 배포해야 하는 단점이 있다. 또한 프리미티브 자료형이나 다른 직렬화 객체 몇 개를 전달하려면 늘 Parcelable 클래스를 구현해야 한다.

### Bundle

번들은 대부분의 직렬화 객체를 포함시킬 수 있다. 따라서 Parcelable, Serializable과 같이 클래스 형태(멤버 함수를 포함하지 않은)가 아닌 데이터 전달이 목적이려면 가장 편리한 클래스다. 만일 클래스 형태를 객체화한 데이터를 전달하려면 속도가 빠른 Parcelable을 이용하자.

### Intent

인텐트는 내부에 번들 객체를 가지기 때문에 Bundle의 장점을 모두 가진다. 또한 앱과 액티비티 매니저, 패키지 매니저 등의 시스템 서비스 간에 약속된 의미 있는 정보를 포함할 수 있다.

3.

인텐트

- 인텐트를 이용해서 원하는 컴포넌트를 실행하기 위해서는 명시적<sup>Explicit</sup>인 방법과 암시적<sup>Implicit</sup>인 방법이 있다.

먼저 명시적인 방법과 이를 위해 인텐트의 어떤 정보가 사용되는지 살펴보자.

참고로 명시적인 방법을 명시적 인텐트라 하고 암시적인 방법을 암시적 인텐트라 한다.

## 명시적 인텐트

정보 분류	멤버변수	설명
컴포넌트 정보	String mPackage ComponentName mComponent	명시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
액션	String mAction	암시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
카테고리	HashSet<String> mCategories	
데이터 위치와 타입	Uri mData String mType	
엑스트라	Bundle mExtras	각종 컴포넌트에 순수 데이터를 전달하기 위한 용도
플래그	int mFlags	각종 컴포넌트를 제어하기 위한 플래그

- 명시적 인텐트는 실행할 액티비티 컴포넌트를 정확히 명시하는 것을 말한다.  
그러므로 실행될 **컴포넌트명**을 인텐트에 작성해주어야 한다.

참고로 명시적 인텐트를 위해 사용된 ComponentName 클래스도 Parcelable을 상속받은 직렬화 클래스다.

### ■ 이전 예제에서 사용한 내용을 잠시 살펴보자.

#### src/AActivity.java

```
...
Intent intent = new Intent();
ComponentName componentName = new ComponentName (
    "com.superdroid.test.activity.b",
    "com.superdroid.test.activity.b.BActivity" );

intent.setComponent( componentName );
startActivity( intent );
...
```

외부 패키지의 액티비티를 실행하기 위해 ComponentName 클래스를 이용하여 실행될 패키지명과 컴포넌트명을 정확히 기록하고 있다.

**하지만 외부 패키지에 있는 액티비티를 활성화하기 위해서는 명시적 인텐트를 사용하는 경우가 거의 없다.**

1. 외부 패키지는 정확한 패키지명과 컴포넌트명을 미리 알고 있는 경우가 드물다.
2. 알고 있더라도 해당 단말기에 실행될 앱이 설치되어 있지 않을 수도 있다.
3. 외부 앱들도 보안상 자신의 액티비티를 외부에 공개하지 않는다.

따라서 명시적 인텐트는 패키지 내부의 액티비티를 실행할 때만 사용하는 것이 맞다.

외부 패키지의 액티비티는 암시적 인텐트를 사용해야 한다.

- 명시적 인텐트로 내부 액티비티를 실행해보자. A 패키지에 A2 액티비티를 추가한다.

### res/layout/activity\_a2\_layout.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="내부 A2 액티비티가 실행 실행되었습니다."/>

</LinearLayout>
```

### src/A2Activity.java

```
public class A2Activity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_a2_layout );
    }
}
```

## 명시적 인텐트

■ 명시적 인텐트로 내부 액티비티를 실행해보자. A 패키지에 A2 액티비티를 추가한다.

### res/layout/activity\_a

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="10dp">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="내부 A2 액티비티 실행" />

</LinearLayout>
```

### src/A2Activity.java

```
public class A2Activity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_a2);
    }
}
```

### AndroidManifest.xml



```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.superdroid.test"
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">

    <activity
        android:name="com.superdroid.test.activity.a.AActivity"
        android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <!-- 호출될 내부 액티비티를 하나 추가한다. -->
    <activity android:name=".A2Activity"
        android:label="A2 Activity">
    </activity>

</manifest>
```

### ■ A 패키지 A1 액티비티에서 A2 액티비티를 실행한다.

src/AActivity.java

```
public class AActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_a_layout );
    }

    public void onClick( View v )
    {
        // ① 첫 번째 방법
        // =====
        /* Intent intent = new Intent();
        ComponentName componentName = new ComponentName (
            "com.superdroid.test.activity.a",
            "com.superdroid.test.activity.a.A2Activity" );

        intent.setComponent( componentName );

        startActivity( intent ); */
        // =====

        // ② 두 번째 방법
        // =====
        startActivity(new Intent( this, A2Activity.class ) );
        // =====
    }
}
```

사실 같은 방법이다. 내부적으로 첫 번째 방법을 사용하기 때문이다.



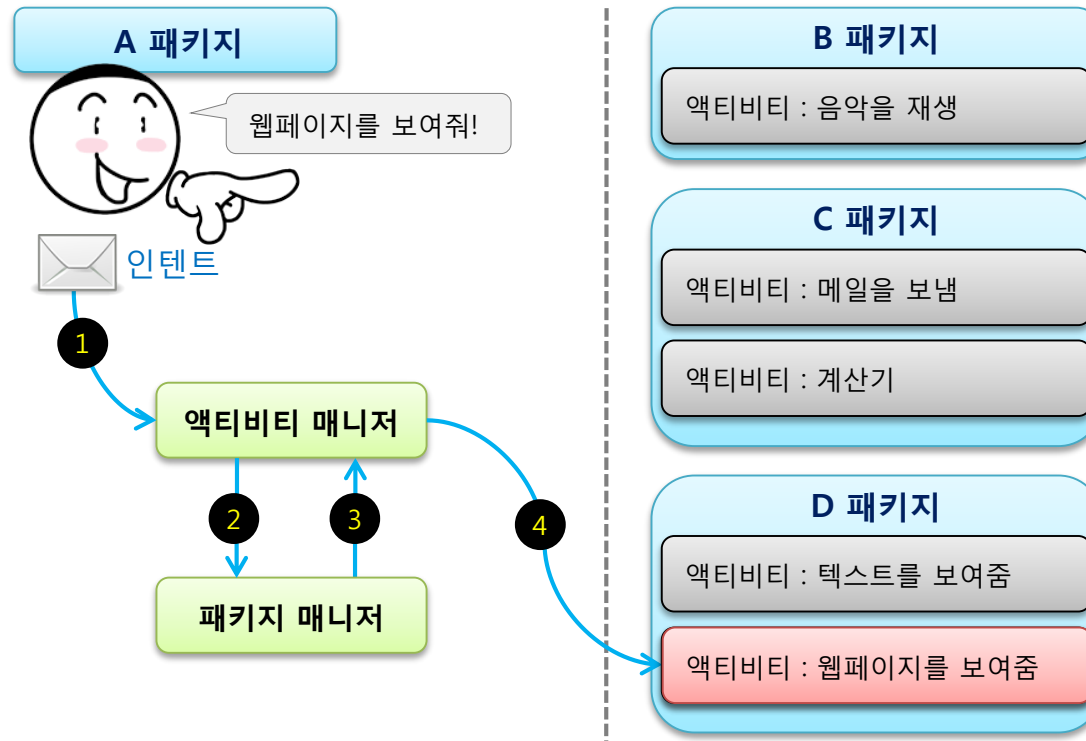
## 암시적 인텐트

정보 분류	멤버변수	설명
컴포넌트 정보	String mPackage ComponentName mComponent	명시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
액션	String mAction	암시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
카테고리	HashSet<String> mCategories	
데이터 위치와 타입	Uri mData String mType	
엑스트라	Bundle mExtras	각종 컴포넌트에 순수 데이터를 전달하기 위한 용도
플래그	int mFlags	각종 컴포넌트를 제어하기 위한 플래그

■ 인텐트를 한국어로 번역하면 '의도'라고 해석할 수 있으며,

암시적Implicit 인텐트에 가장 부합되는 의미이기도 하다.

이는 명시적 인텐트와 달리 암시적 인텐트는 어떤 '의도'만으로 원하는 컴포넌트를 실행할 수 있기 때문이다.



### ■ 의도 정보는 인텐트의 네 가지 정보를 통해 설정한다.

① **액션** : 동작을 설명하는 미리 정의된 문자열을 말한다.

예를 들어 SMS를 발송한다, 메일을 보낸다, 전화를 건다, 이미지를 본다 등이 될 수 있다.

② **카테고리** : 액티비티의 분류에 해당한다.

예를 들어 해당 액티비티가 홈의 앱 아이콘을 눌렀을 때 실행되는 액티비티라면,  
android.intent.category.LAUNCHER라 한다. 따라서 대부분 앱은  
android.intent.category.LAUNCHER 카테고리를 가진 액티비티 하나가 존재한다.

③ **데이터 위치** : 실행될 컴포넌트가 특정 데이터를 필요로 한다면 추가될 수 있다.

예를 들어 액션이 음악을 재생한다면 데이터는 음악 파일의 경로가 될 수 있다.

④ **데이터 타입** : ③번의 데이터 타입을 정의할 수 있다.

예를 들어 데이터가 음악 파일의 주소를 넘겨준다면 타입은 MP3, WAV, MOV 등과  
같이 음악 파일의 다양한 포맷을 추가할 수 있다.

## 암시적 인텐트 – 액션과 카테고리

정보 분류	멤버변수	설명
컴포넌트 정보	String mPackage ComponentName mComponent	명시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
액션	String mAction	암시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
카테고리	HashSet<String> mCategories	
데이터 위치와 타입	Uri mData String mType	
엑스트라	Bundle mExtras	각종 컴포넌트에 순수 데이터를 전달하기 위한 용도
플래그	int mFlags	각종 컴포넌트를 제어하기 위한 플래그

### ■ 액션과 카테고리를 이용하여 이메일 열기.

#### src/MainActivity.java

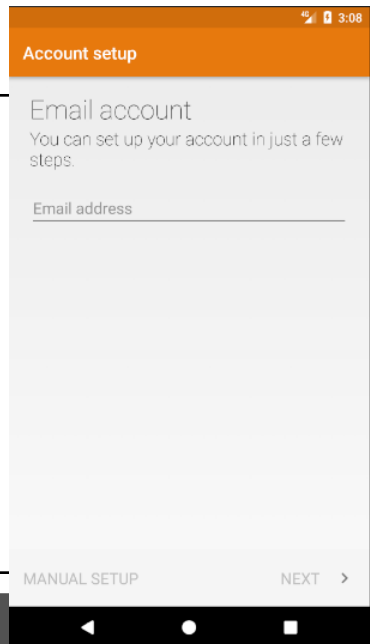
```

...
// ① 단말기에 설치된 앱을 실행했을 때 가장 첫 번째로 실행되는 액티비티를 보여달라는
// 액션
intent.setAction( Intent.ACTION_MAIN );

// ② 안드로이드 기본 앱 중 이메일 카테고리
intent.addCategory( Intent.CATEGORY_APP_EMAIL );

startActivity(Intent.createChooser(intent, "hi"));

```



## 암시적 인텐트 – 액션과 데이터 위치

정보 분류	멤버변수	설명
컴포넌트 정보	String mPackage ComponentName mComponent	명시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
액션	String mAction	암시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
카테고리	HashSet<String> mCategories	
데이터 위치와 타입	Uri mData String mType	
엑스트라	Bundle mExtras	각종 컴포넌트에 순수 데이터를 전달하기 위한 용도
플래그	int mFlags	각종 컴포넌트를 제어하기 위한 플래그

■ 액션과 데이터 위치를 이용하여 외부 브라우저 컴포넌트를 실행해보자.

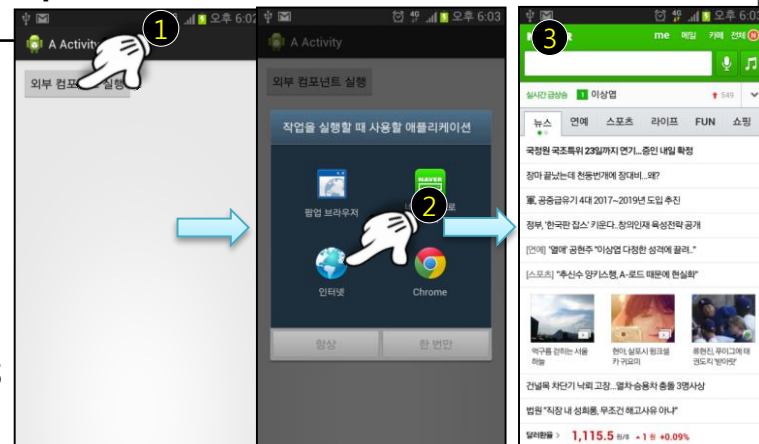
src/AActivity.java

```

...
// ① 어떤 데이터를 보여달라는 액션
intent.setAction( Intent.ACTION_VIEW );

// ② 다음 위치의 데이터를 보여달라는 의미
intent.setData( Uri.parse( "http://m.naver.com" ) );
...

```

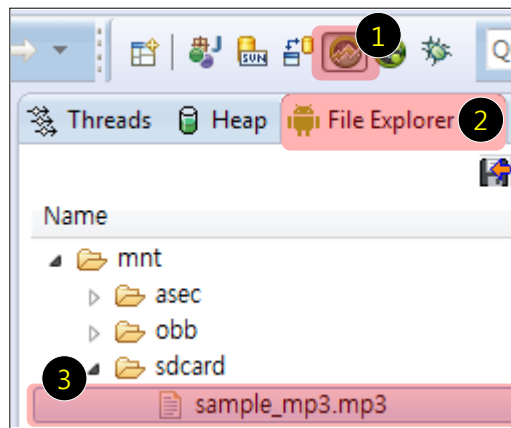


## 암시적 인텐트 – 액션과 데이터 위치 및 타입

정보 분류	멤버변수	설명
컴포넌트 정보	String mPackage ComponentName mComponent	명시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
액션	String mAction	암시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
카테고리	HashSet<String> mCategories	
데이터 위치와 타입	Uri mData String mType	
엑스트라	Bundle mExtras	각종 컴포넌트에 순수 데이터를 전달하기 위한 용도
플래그	int mFlags	각종 컴포넌트를 제어하기 위한 플래그

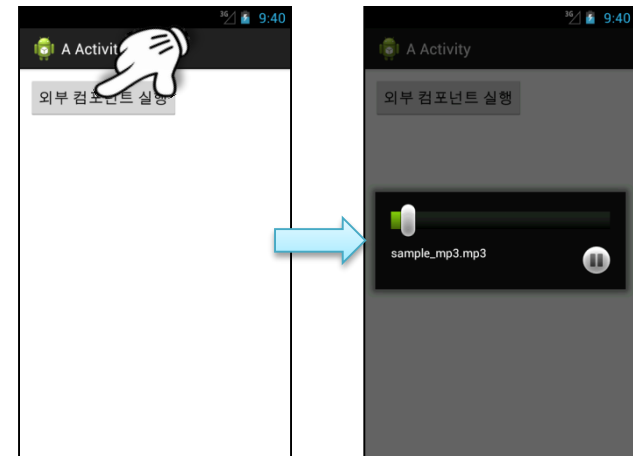
■ 액션과 데이터 위치를 이용하여 외부 브라우저 컴포넌트를 실행해보자.

SD 카드로 음악 파일 복사하기



### src/AActivity.java

```
...  
// ① 어떤 데이터를 보여달라는 액션  
intent.setAction( Intent.ACTION_VIEW );  
  
// ② 다음 위치의 데이터를 보여달라는 의미  
String audioPath = "file:/// " +  
    Environment.getExternalStorageDirectory() +  
    "/sample_mp3.mp3";  
  
// ③ 모든 오디오 포맷의 데이터 타입  
intent.setDataAndType( Uri.parse( audioPath ),  
    "audio/*");  
...
```



## 암시적 컴포넌트 등록 -

### 액션과 카테고리로 실행 가능한 암시적 컴포넌트 만들기

■ 앞서 암시적 인텐트를 이용하여 외부 컴포넌트 실행 방법을 알아보았다.

이제부터 직접 액티비티 컴포넌트를 구현하고 외부 앱들이 사용할 수 있도록 등록해보자.

B 앱을 수정하여 암시적 컴포넌트 등록 방법을 알아보자.



## 암시적 컴포넌트 등록 -

### 액션과 카테고리로 실행 가능한 암시적 컴포넌트 만들기

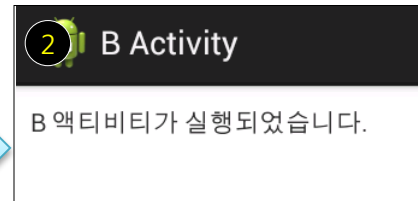
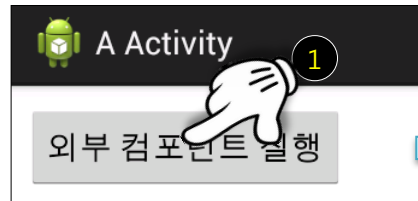
#### AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=...
    ...
    <!-- 암시적 액티비티 컴포넌트를 등록한다. -->
    <activity
        android:name="com.superdroid.test.activity.b.BActivity"
        android:label="@string/app_name" >

        <intent-filter>
            <action android:name="action.ACTION_IMAGE_VIEW" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>

    </activity>
</application>
</manifest>
```

B 앱에 암시적  
액티비티 등록



#### src/AActivity.java

```
...
// 이미지를 보여달라는 액션
intent.setAction( "action.ACTION_IMAGE_VIEW" );
...
```

A 앱 액티비티에서  
B 앱의 암시적  
액티비티 실행

## 암시적 컴포넌트 등록 -

### 액션과 카테고리로 실행 가능한 암시적 컴포넌트 만들기

■ 주의 : `<category android:name="android.intent.category.DEFAULT" />`

• 액티비티 컴포넌트 등록 필터 정보

```
<action android:name="action.ACTION_IMAGE_VIEW" />
```

```
<category android:name="android.intent.category.DEFAULT" />
```

• 인텐트

```
intent.setAction( "action.ACTION_IMAGE_VIEW" );
```



```
<category android:name="android.intent.category.DEFAULT" />
```

안드로이드 시스템은 이 카테고리를 강제로 넣어 버린다.

따라서 인텐트 필터에  
카테고리 DEFAULT를  
설정하지 않으면 어떤  
암시적 인텐트도  
받을 수 없다.

그러므로 DEFAULT는 암시적 인텐트 사용을 위한 카테고리이기도 하다.

## 암시적 컴포넌트 등록 -

## 액션과 카테고리로 실행 가능한 암시적 컴포넌트 만들기

■ 참고 : `<category android:name="android.intent.category.BROWSABLE"/>`



```
<activity
    android:name="com.superdroid.test.activity.b.BActivity"
    android:label="@string/app_name" >

    <intent-filter>


        <action android:name="action.ACTION_IMAGE_VIEW" />
        <category android:name="android.intent.category.DEFAULT" />

        브라우저에서 실행 가능한 액티비티라는 카테고리
        <category android:name="android.intent.category.BROWSABLE"/>

        <data android:scheme="superdroid" />

    </intent-filter>

</activity>
```

 B Activity

B 액티비티가 실행되었습니다.

링크 : 안드로이드 앱이름://원하는 기능?전달 인자

예) `superdroid://list?age=18...`

## 암시적 컴포넌트 등록 -

### 액션과 카테고리로 실행 가능한 암시적 컴포넌트 만들기

#### ■ 액션과 카테고리명을 지정할 때 주의사항

앞서 액션명으로 `action.ACTION_IMAGE_VIEW`를 사용했다. 하지만 이렇게 자신만이 알 수 있는 이름을 사용해서는 외부 패키지에서 해당 컴포넌트를 실행할 수 없다. 그러므로 안드로이드에서 정의한 `android.intent.action.VIEW`를 사용하고 데이터와 타입을 이용해서 이미지 파일의 경로 및 이미지 포맷을 설정해야 한다. 물론 카테고리도 마찬가지다. 예제는 암시적 컴포넌트 등록의 이해를 돕고자 작성된 것이다.

<intent-filter>

```
<action android:name="action.ACTION_VIEW" />
```

어떤 콘텐츠를 보여 준다는 액션

```
<category android:name="android.intent.category.DEFAULT" />
```

암시적 인텐트 허용

```
<category android:name="android.intent.category.BROWSABLE"/>
```

브라우저에서 실행될 수 있다

```
...
```

기타 카테고리가 많이 지원될 수록  
외부에서 실행될 확률이 높겠다.

</intent-filter>

참고로 안드로이드에서 정의한 액션들은 안드로이드 개발자 사이트를 참조하자.

<http://developer.android.com/reference/android/content/Intent.html>

## 암시적 컴포넌트 등록 – 액션과 카테고리 그리고 데이터 위치 및 타입으로 실행 가능한 암시적 컴포넌트 만들기

■ 데이터 위치 인텐트 필터에 대해서 배우려면 먼저 URI<sup>Uniform Resource Identifiers</sup>에 대한 이해가 필요하다.

이미지를 보여주는 컴포넌트를 실행하려면 실제 이미지 데이터를 전달해야 할 것이다.

하지만 이미지 데이터는 용량이 크고 다른 앱 프로세스에게 전달하려면 직렬화 객체를 사용해야 할 것이다.

또한 이미지 데이터 자체를 전달하는 것은 리소스 낭비가 너무 심하고 성능이 좋지 않다.

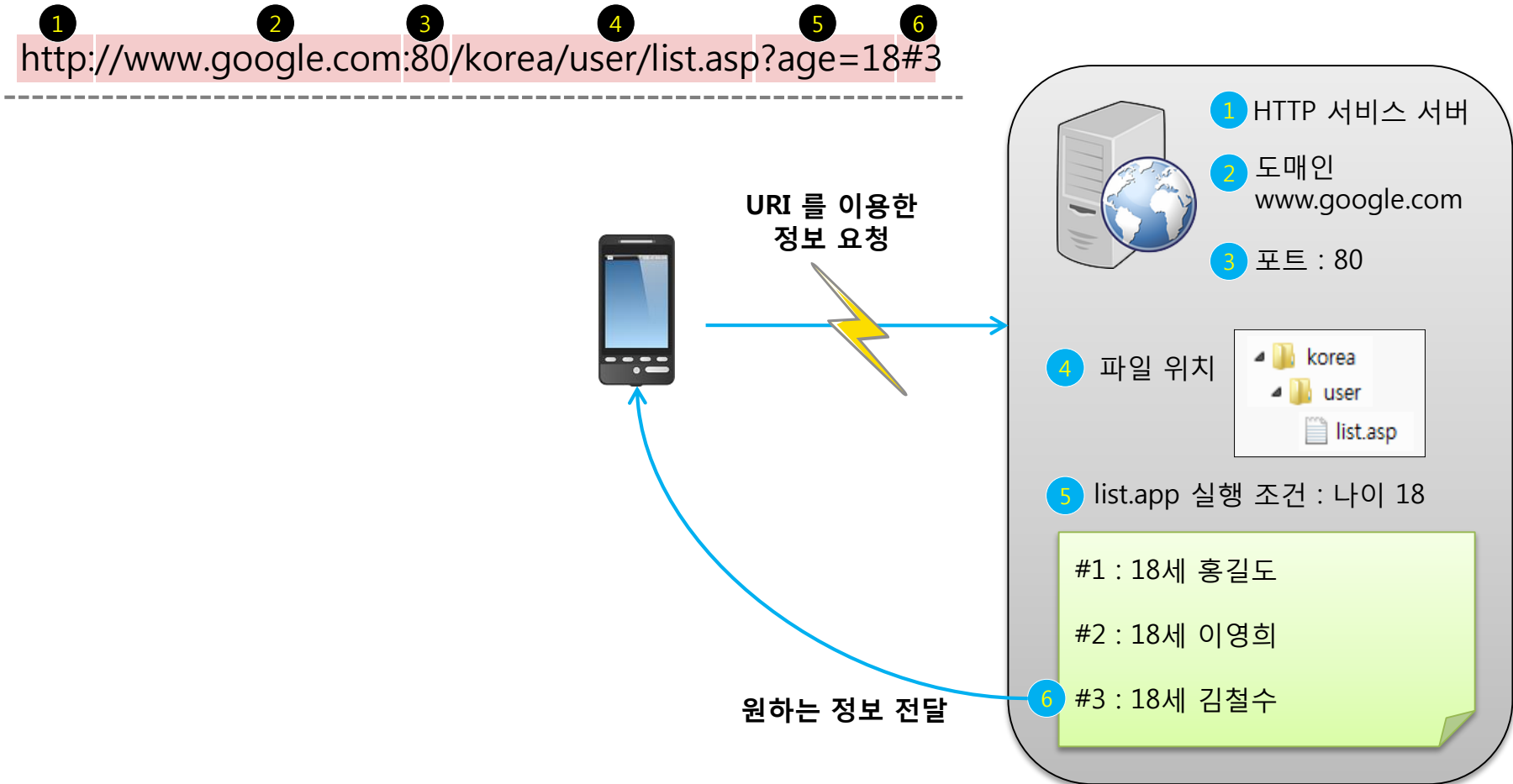
따라서 안드로이드는 인텐트를 통해 데이터를 전달할 때 실제 데이터 자체를 전달하지 않고 데이터가 존재하는 위치 정보만 전달한다.

여기서 데이터 위치를 표현할 때 URI 형식을 사용한다. URI는 RFC2396에서 정의하는 규약이며, 상세한 규약 설명은 다음의 링크 주소를 참조하자.

<http://www.ietf.org/rfc/rfc2396.txt?number=2396>

## 암시적 컴포넌트 등록 - 액션과 카테고리 그리고 데이터 위치 및 타입으로 실행 가능한 암시적 컴포넌트 만들기

URI 형식은 위치 정보를 표현하기에 매우 적합한 구조다. 참고로 브라우저를 통해 구글 웹페이지를 호출할 때 사용하는 `http://www.google.com` URL도 URI의 서브셋이다. 그렇다면 URI 형식이 데이터 위치 정보를 얼마나 효율적으로 표현할 수 있는지 살펴보자.



## 암시적 컴포넌트 등록 – 액션과 카테고리 그리고 데이터 위치 및 타입으로 실행 가능한 암시적 컴포넌트 만들기

Fragment

1 Scheme   2 Host   3 Port   4 Path   5 Query   6

http://www.google.com:80/korea/user/list.asp?age=18#3

7 Authority   8 Last Path Segment

http://www.google.com:80/korea/user/list.asp?age=18#3

- 안드로이드는 데이터 위치를 설정할 때 Uri 규약을 사용하고, 내부적으로 편리한 Uri 클래스도 제공한다. 예제를 통해 살펴보자.

# 암시적 컴포넌트 등록 – 액션과 카테고리 그리고 데이터 위치 및 타입으로 실행 가능한 암시적 컴포넌트 만들기

## src/MainActivity.java

```
intent.setAction( "action.ACTION_IMAGE_VIEW" );
```

```
Uri imageUri = Uri.parse(
    "http://www.superdroid.com:80/files/images/test.png?a=0#3" );
```

```
intent.setDataAndType( imageUri, "image/png" );
```

Scheme

Host

Port

Path

http://www.google.com:80/korea/user/list.asp?age=18#3

## AndroidManifest.xml

...

<!-- 암시적 액티비티 컴포넌트를 등록한다. -->

<activity

android:name="com.superdroid.test.activity.b.BActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="action.ACTION\_IMAGE\_VIEW" />

<category android:name="android.intent.category.DEFAULT" />

<data android:scheme="http"

android:host="www.superdroid.com"

android:port="80"

android:path="/files/images/test.png"

android:mimeType="image/png"/>

</intent-filter>

</activity>

...



# 암시적 컴포넌트 등록 – 액션과 카테고리 그리고 데이터 위치 및 타입으로 실행 가능한 암시적 컴포넌트 만들기

8#3

src/BActivity.java

```
public class BActivity extends Activity
{
    @Override
    protected void onCreate( Bundle savedInstanceState )
    {
        ...

        Intent intent = getIntent();

        // ① 전달 받은 인텐트에서 데이터를 추출한다.
        Uri imageUri = intent.getData();

        // ② 데이터 URI의 내용을 문자열로 저장한다.
        StringBuilder temp = new StringBuilder();
        temp.append( "Scheme : " + imageUri.getScheme() + "\n" );
        temp.append( "Host : " + imageUri.getHost() + "\n" );
        temp.append( "Port : " + imageUri.getPort() + "\n" );
        temp.append( "Path : " + imageUri.getPath() + "\n" );
        temp.append( "Query : " + imageUri.getQuery() + "\n" );
        temp.append( "Fragment : " + imageUri.getFragment() + "\n" );
        temp.append( "Authority : " + imageUri.getAuthority() + "\n" );
        temp.append( "Last Path Segment : " + imageUri.getLastPathSegment() );

        // ③ 텍스트뷰에 데이터 URI 문자열을 출력한다.
        TextView textView = (TextView) findViewById( R.id.intent_received_data );
        textView.setText( temp );
    }
}
```

T" />

ng"

# 암시적 컴포넌트 등록 - 액션과 카테고리 그리고 데이터 위치 및 타입으로 실행 가능한 암시적 컴포넌트 만들기

src/BActivity.java

```
public class BActivity extends Activity
{
```

```
    @Override
    protected void
```

```
    {
```

```
        Intent
```

```
        // ① 전
        Uri ima
```

```
        // ② 데
        StringB
        temp.ap
        temp.ap
        temp.ap
        temp.ap
        temp.ap
        temp.ap
        temp.ap
        temp.ap
```

```
        // ③ 텍
        TextVie
        textVie
```

```
    }
```

```
}
```

인텐트 발신 측 데이터 URI

```
Uri imageUri = Uri.parse(
    "http://www.superdroid.com:80/files/images/test.png?a=0#3" );
```

```
intent.setDataAndType( imageUri, "image/png" );
```

암시적 컴포넌트 인텐트 필터 설정 (AndroidManifest.xml)

<data

```
    android:scheme="http"
    android:host="www.superdroid.com"
    android:port="80"
    android:path="/files/images/test.png"

    android:mimeType="image/png"/>
```

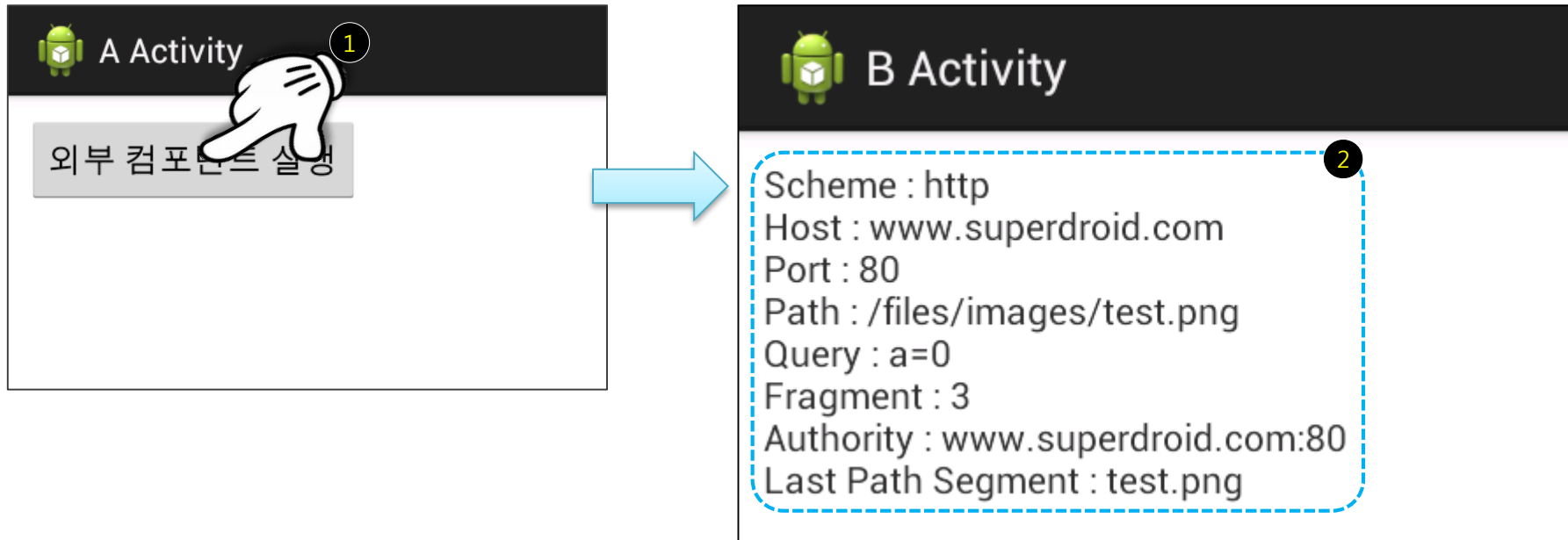
단 하나의 조건이라도 일치하지 않으면 액티비티는 실행되지 않는다.

8#3

T" />

ng"

# 암시적 컴포넌트 등록 - 액션과 카테고리 그리고 데이터 위치 및 타입으로 실행 가능한 암시적 컴포넌트 만들기



1 Scheme    2 Host    3 Port    4 Path    5 Query    6 Fragment  
<http://www.google.com:80/korea/user/list.asp?age=18#3>

7 Authority    8 Last Path Segment  
<http://www.google.com:80/korea/user/list.asp?age=18#3>

## 암시적 컴포넌트 등록 – 액션과 카테고리 그리고 데이터 위치 및 타입으로 실행 가능한 암시적 컴포넌트 만들기

### ■ Uri에서 데이터 경로를 대체하는 유연한 속성들

AndroidManifest.xml

...

```
<data android:scheme="http"
      android:host="www.superdroid.com"
      android:port="80"
      android:path="/files/images/test.png"
      android:mimeType="image/png"/>
```

...

- pathPrefix : 경로의 앞부분 문자열을 고정하고 뒤에 오는 문자열은 무엇이든 허용한다.  
예를 들어 pathPrefix를 /aaa라고 설명하면 /aaa/bbb/cc/a.png든 /aaa/a.png든 모두 허용할 수 있다.
- pathPattern : pathPrefix와 유사하나 pathPattern은 뒤에 문자열만 가변을 허용하는 것이 아니라  
.\* 문자열이 들어간 어느 구간이나 가변을 허용하게 할 수 있다.  
예를 들어 pathPattern을 /.\*/a.png라고 설정하면 /aaa/a.png든 /bbb/a.png든 모두 허용될 수 있다.



## 데이터 타입 MIME(Multipurpose Internet Mail Extensions)

데이터 타입을 일반적으로 마임<sup>MIME</sup>이라 부르는데 오래 전 이메일 서비스에서 메일에 첨부된 파일의 데이터 타입을 구분하기 위한 용도로 생겨났다. 사용 예를 살펴보자.

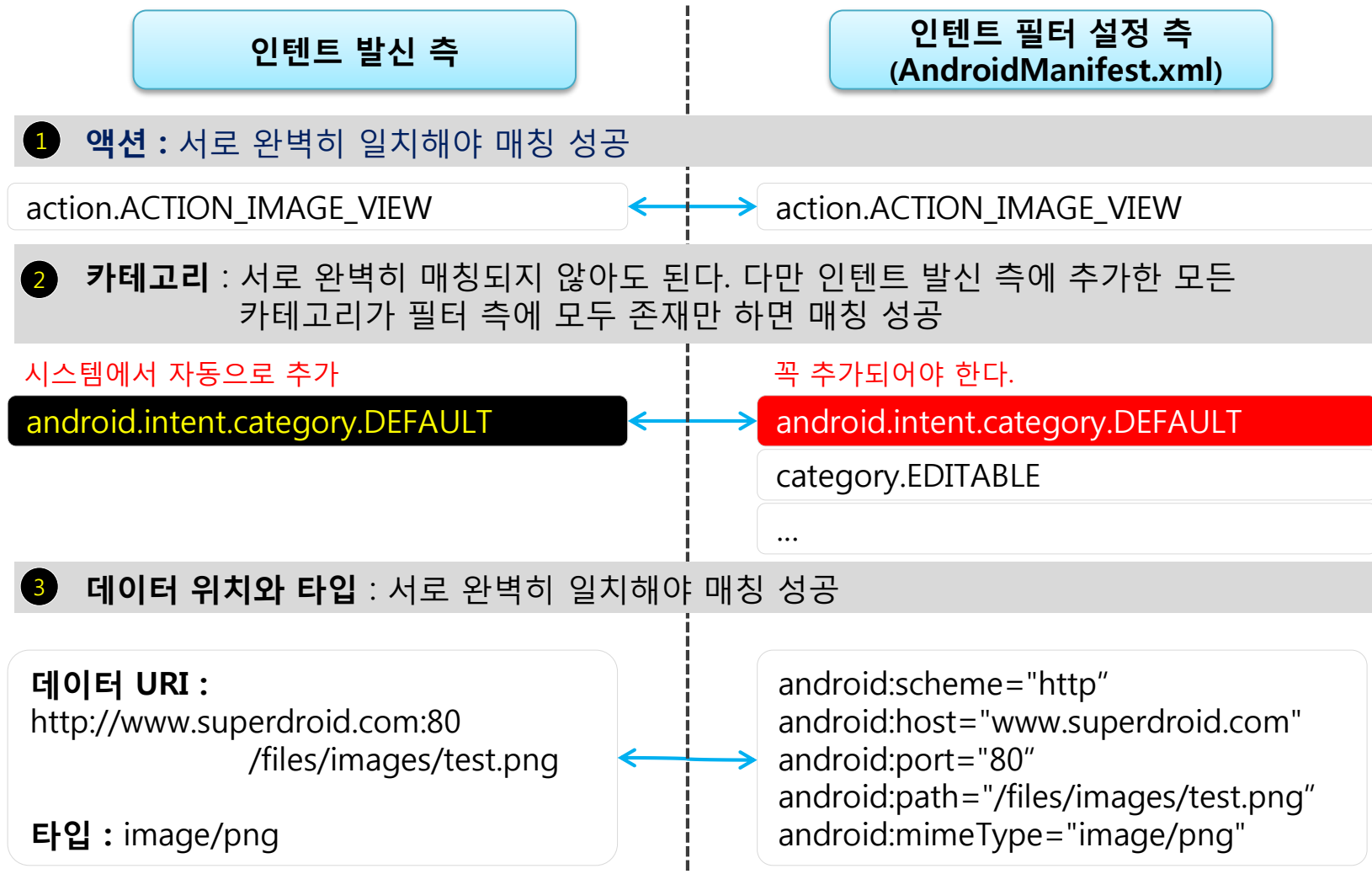
- text/html : 텍스트 데이터며, 텍스트 내용은 HTML 형태 파일이다.
- video/mpeg : 비디오 데이터며, MPEG 영상 형태의 파일이다.
- image/jpeg : 이미지 데이터며, JPEG 형태의 파일이다.
- audio/x-wav : 오디오 데이터며, wav 형태의 파일이다.

<type>/<sub type> 형식으로 쉽게 데이터 타입을 구분할 수 있다.

이러한 장점으로 대부분의 데이터 타입을 표현할 때, 마임을 사용하고 있다.

## 암시적 인텐트와 암시적 컴포넌트 인텐트 필터 매칭 조건 정리

- 인텐트 필터를 작성하고 그에 매칭되는 인텐트를 만드는 것이 다소 복잡해보일 수 있으므로 아래의 그림을 통해 다시 한 번 정리해보자.



정보 분류	멤버변수	설명
컴포넌트 정보	String mPackage ComponentName mComponent	명시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
액션	String mAction	암시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
카테고리	HashSet<String> mCategories	
데이터 위치와 타입	Uri mData String mType	
엑스트라	Bundle mExtras	각종 컴포넌트에 순수 데이터를 전달하기 위한 용도
플래그	int mFlags	각종 컴포넌트를 제어하기 위한 플래그

■ 엑스트Extra라는 실행되는 액티비티에 전달할 순수 데이터다.

따라서 실행될 액티비티 조건과는 상관없고 명시적, 암시적 인텐트 모두 사용할 수 있다.

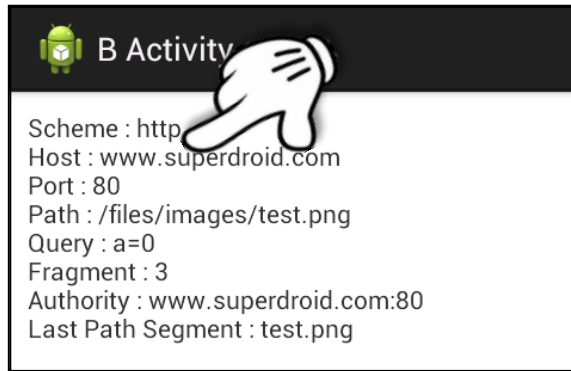
■ 엑스트라의 정보 형태는 번들이다. 따라서 각종 직렬화된 객체라면 자유롭게 추가할 수 있다.

엑스트라에 대해서는 예제를 충분히 실습했기 때문에 추가 설명은 생략한다.

정보 분류	멤버변수	설명
컴포넌트 정보	String mPackage ComponentName mComponent	명시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
액션	String mAction	암시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
카테고리	HashSet<String> mCategories	
데이터 위치와 타입	Uri mData String mType	
엑스트라	Bundle mExtras	각종 컴포넌트에 순수 데이터를 전달하기 위한 용도
플래그	int mFlags	각종 컴포넌트를 제어하기 위한 플래그

- 인텐트 플래그는 안드로이드의 컴포넌트들을 제어하거나 상태를 변경하는 등 다양한 목적으로 사용된다. 하지만 플래그에 대해서는 안드로이드 컴포넌트를 대부분 이해해야 설명이 가능하다. 따라서 앞으로 배울 모든 장에 걸쳐 설명한다. 단 플래그의 이해를 위해 액티비티 컴포넌트와 관련된 플래그 한 가지만 알아보자.





- **Intent.FLAG\_ACTIVITY\_NO\_ANIMATION** : 액티비티가 실행될 때 동작하는 애니메이션 효과를 사용하지 않는다.

### src/MainActivity.java

```
public void onClick( View v )  
{  
    ...  
    // 액티비티 구동 애니메이션을 막는 인텐트 플래그 설정  
    intent.addFlags( Intent.FLAG_ACTIVITY_NO_ANIMATION );  
}
```

## 인텐트 패키지

정보 분류	멤버변수	설명
컴포넌트 정보	<code>String mPackage</code> <code>ComponentName mComponent</code>	명시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
액션	<code>String mAction</code>	암시적으로 원하는 컴포넌트(액티비티, 서비스, 리시버)를 실행하기 위한 용도
카테고리	<code>HashSet&lt;String&gt; mCategories</code>	
데이터 위치와 타입	<code>Uri mData</code> <code>String mType</code>	
엑스트라	<code>Bundle mExtras</code>	각종 컴포넌트에 순수 데이터를 전달하기 위한 용도
플래그	<code>int mFlags</code>	각종 컴포넌트를 제어하기 위한 플래그

■ 암시적 인텐트는 불특정 다수의 패키지를 대상으로 액션, 카테고리 등만 일치하면 해당 컴포넌트를 실행할 수 있다. 만일 일치하는 컴포넌트들이 포함된 패키지가 둘 이상 존재하는 경우, 사용자가 선택할 수 있는 시스템 팝업이 구동된다.

■ 하지만 여기서 분명 특정 패키지에 존재하는 컴포넌트만을 실행하고 싶을 때가 있을 것이다. 물론 명시적 인텐트를 사용하면 가능하겠지만, 액션과 카테고리 등을 사용해야 하는 암시적 인텐트는 방법이 없다. 이를 위해 안드로이드는 인텐트 패키지를 제공하고 있다. 인텐트 패키지는 원하는 패키지로 한정하여 컴포넌트를 실행할 수 있다.

## 인텐트 패키지

- 인텐트 패키지는 암시적 컴포넌트를 실행할 때 많이 사용된다. 하지만 특정 패키지로 제한하기 때문에 오히려 명시적 인텐트에 가깝다.

src/MainActivity.java

```
intent.setPackage( "com.superdroid.test.activity.b" );
```