



자바응용SW(앱)개발자양성

액티비티 수명주기

백제직업전문학교

김영준 강사

1.

액티비티 생명주기

■ 안드로이드 컴포넌트 중 액티비티는 내/외부적으로 가장 간섭을 많이 받는다.

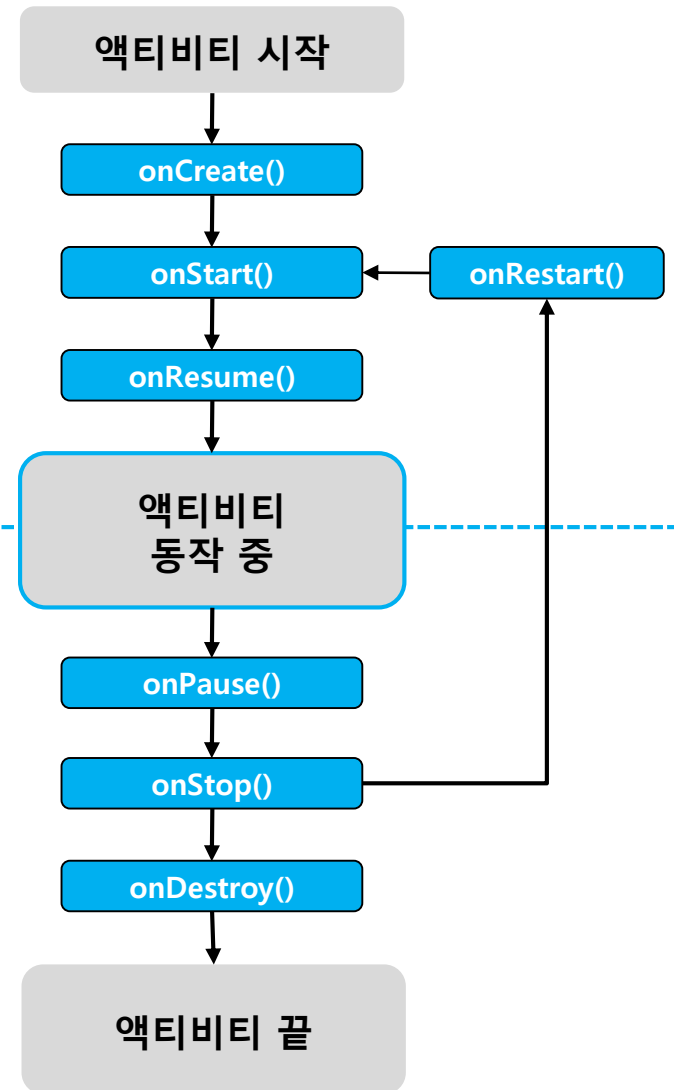
예를 들어 사용자가 액티비티에서 어떤 작업을 하던 중 전화가 오면 화면이 바뀌기도 하고, 단말기를 가로, 세로로 회전하면 화면 사이즈가 바뀌기도 한다. 그러므로 개발자는 액티비티 상태 변화에 따라 적절한 대비를 해야 하고, 어떤 환경에서도 정상적으로 동작시켜야 한다.

하지만 이는 개발자에게 적지 않은 부담일 것이다. 이를 위해 안드로이드는 복잡한 액티비티의 상태 변화를 몇 가지로 분류하고, 그 분류에 따라 개발자에게 대처할 수 있는 몇 가지 함수를 제공한다.

이것이 바로 액티비티 생명주기 함수다.

개발자는 제공되는 함수 내에서 다양한 상태 변화를 대비할 수 있다.

액티비티 생명주기 함수



액티비티 생명주기 함수

src/MainActivity.java

```
public class MainActivity extends Activity
{
    @Override
    protected void onCreate(
        Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );
        setContentView( R.layout.activity_main );
        Log.e( "superdroid", "onCreate()" );
    }

    @Override
    protected void onRestart()
    {
        super.onRestart();
        Log.v( "superdroid", "onRestart()" );
    }

    @Override
    protected void onStart()
    {
        super.onStart();
        Log.d( "superdroid", "onStart()" );
    }
```

```
    @Override
    protected void onResume()
    {
        super.onResume();
        Log.i( "superdroid", "onResume()" );
    }

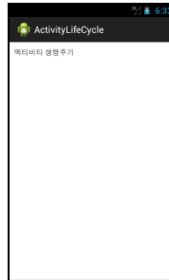
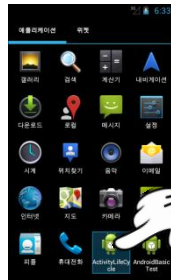
    @Override
    protected void onPause()
    {
        super.onPause();
        Log.i( "superdroid", "onPause()" );
    }

    @Override
    protected void onStop()
    {
        super.onStop();
        Log.d( "superdroid", "onStop()" );
    }

    @Override
    protected void onDestroy()
    {
        super.onDestroy();
        Log.e( "superdroid", "onDestroy()" );
    }
}
```

액티비티 생명주기 함수

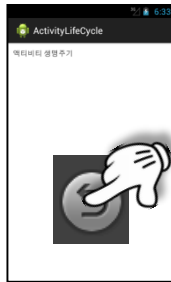
액티비티 실행



● 로그 출력 결과

↓
`onCreate()`
↓
`onStart()`
↓
`onResume()`

액티비티 종료

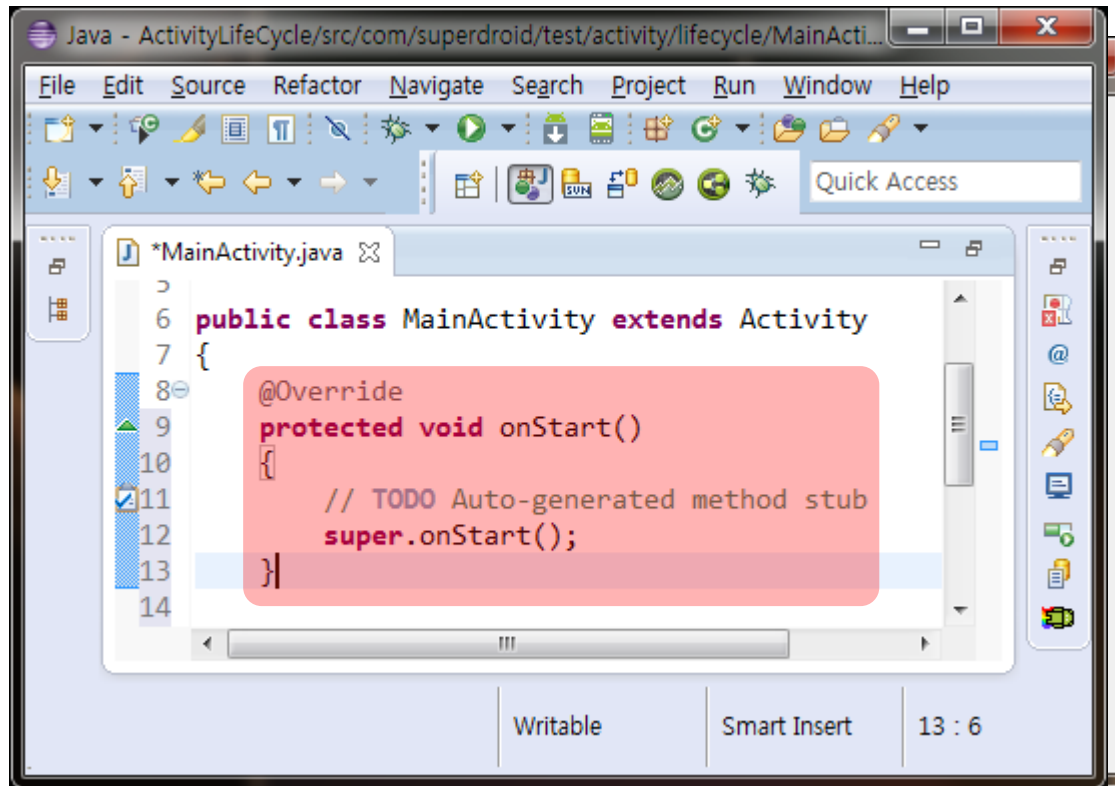


● 로그 출력 결과

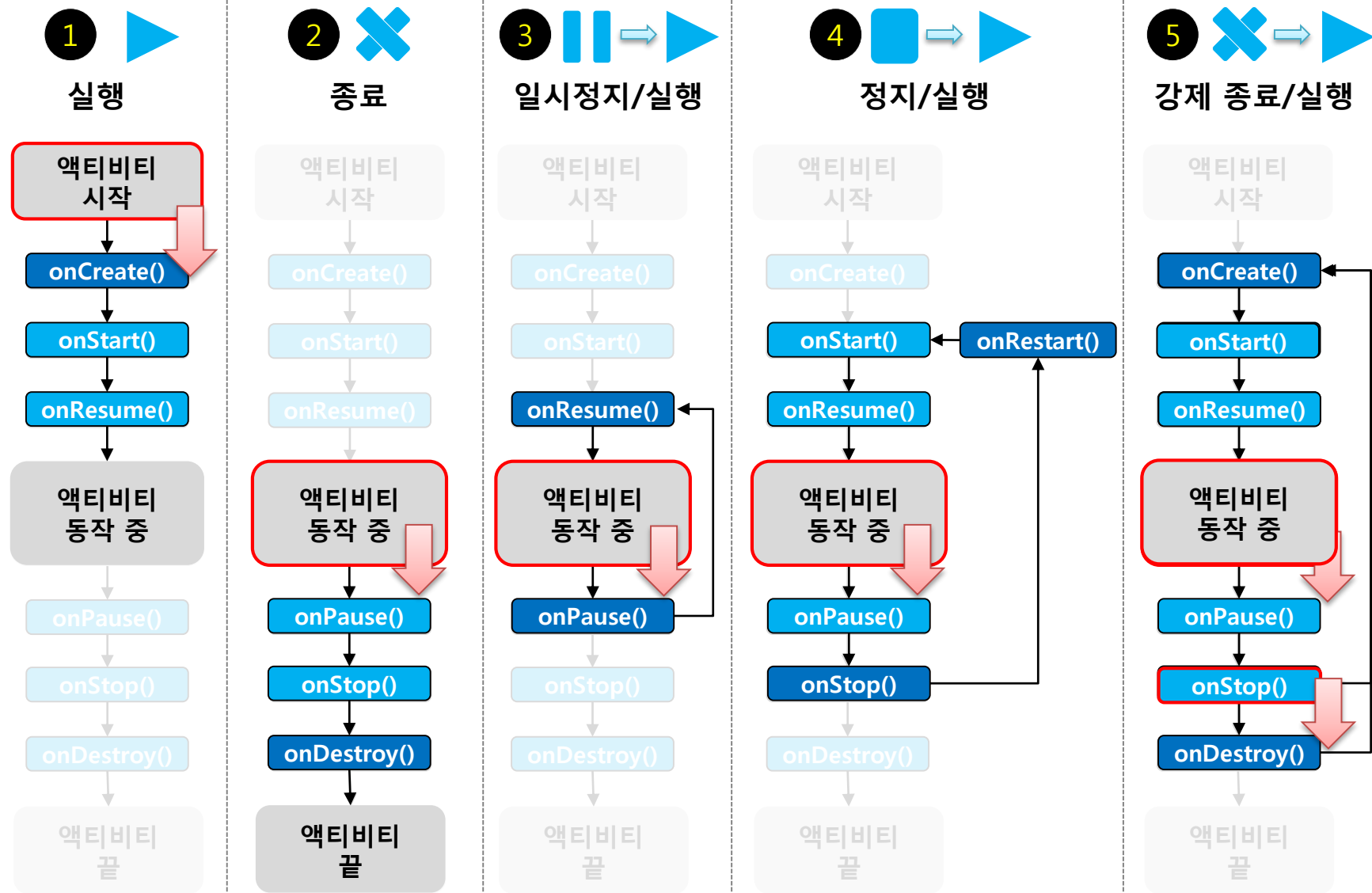
↓
`onCreate()`
↓
`onStart()`
↓
`onResume()`
↓
`onPause()`
↓
`onStop()`
↓
`onDestroy()`

액티비티 생명주기 함수

■ 이클립스에서 손쉽게 재정의의 함수 추가하는 방법



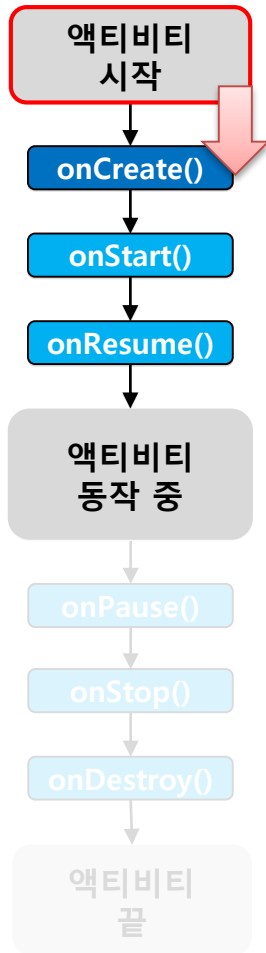
액티비티 상태별 호출되는 생명주기 함수



액티비티 상태 - 액티비티 실행과 종료 상태

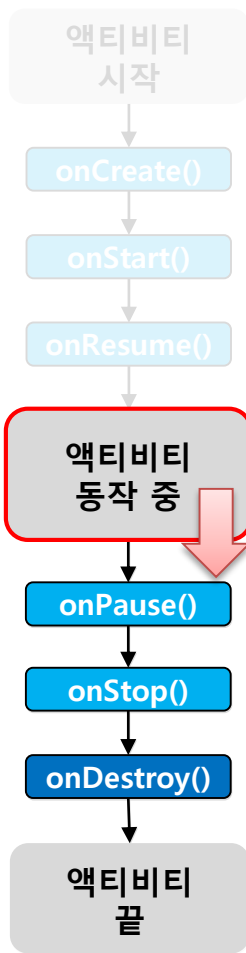
1 ▶

실행



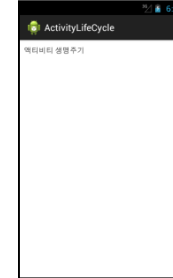
2 ✕

종료



■ 아주 일반적인 상황이다.

액티비티 실행



● 로그 출력 결과

```

onCreate ()
onStart ()
onResume ()
    
```

액티비티 종료



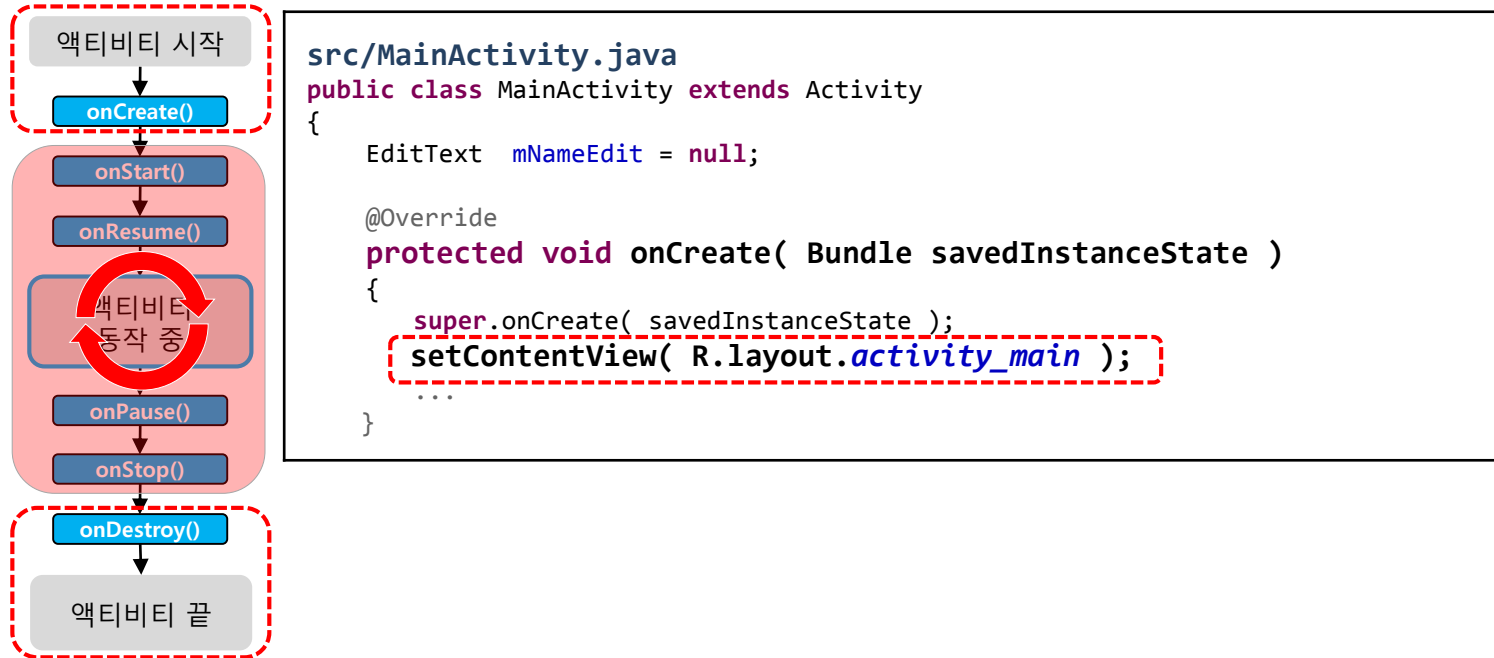
● 로그 출력 결과

```

onCreate ()
onStart ()
onResume ()
onPause ()
onStop ()
onDestroy ()
    
```

액티비티 상태 - 액티비티 실행과 종료 상태

■ onCreate와 짝을 이루는 onDestroy

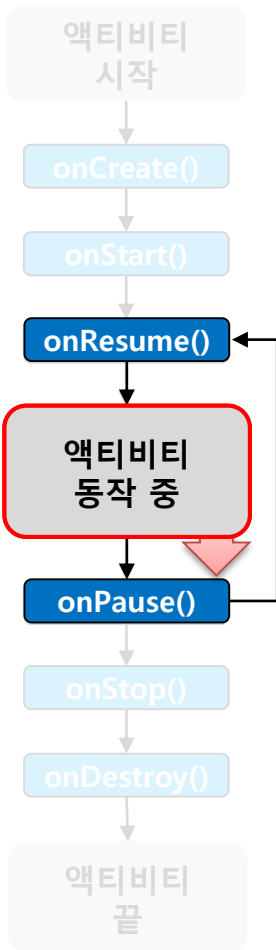


- onCreate는 액티비티의 시작이고, onDestroy는 액티비티의 끝이다.
- 따라서 일반적으로 onCreate에서는 액티비티를 실행하기 위한 객체 생성 및 초기화를 수행하고, onDestroy에서는 사용한 객체를 반환 작업을 수행한다.
- 다른 생명주기들은 그 주기를 반복적으로 실행될 수 있기 때문에 객체 생성 및 초기화 과정을 수행할 수가 없다.

액티비티 상태 - 액티비티 일시 정지와 재실행 상태



일시정지/실행



B 액티비티 뒤로
A 액티비티가 보인다.

액티비티 상태 - 액티비티 일시 정지와 재실행 상태

res/layout/b_activity_layout.xml

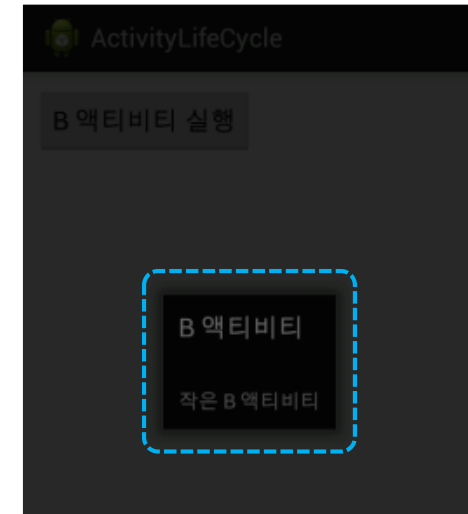
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10dp"
    android:orientation="vertical">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="작은 B 액티비티" />

</LinearLayout>
```

src/BActivity.java

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



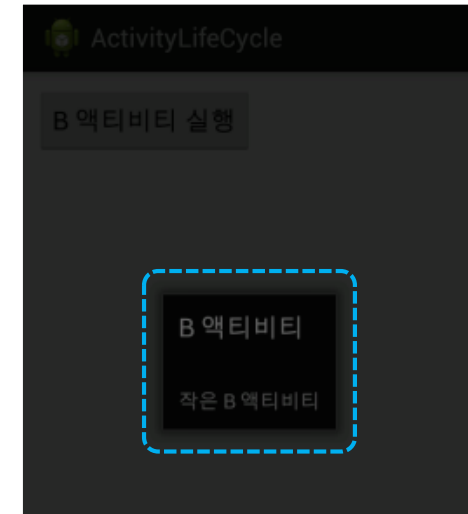
액티비티 상태 - 액티비티 일시 정지와 재실행 상태

res/layout/b_activity_layout.xml

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="작은 B 액티비티" />

</LinearLayout>
```



src/BActivity.java

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

androidManifest.xml

```
...
<activity
    android:name=".BActivity"
    android:label="B 액티비티"
    android:theme="@android:style/Theme.Dialog"/>

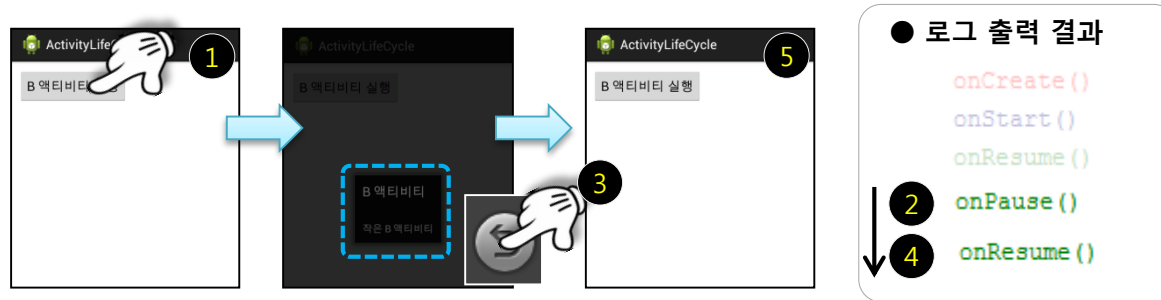
</application>
</manifest>
```

src/MainActivity.java

```
public class MainActivity extends Activity
{
    ...

    public void onClick( View v )
    {
        startActivity( new Intent( this, BActivity.class) );
    }
}
```

액티비티 상태 - 액티비티 일시 정지와 재실행 상태



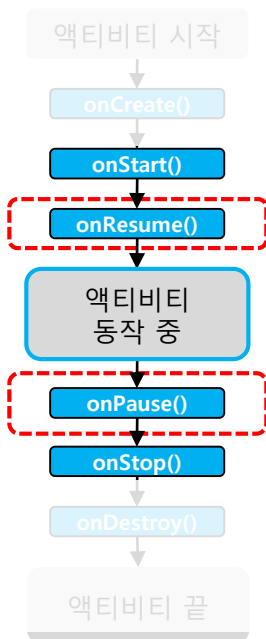
■ onResume과 짝을 이루는 onPause

- onPause는 현재 액티비티가 다른 액티비티로부터 부분적으로 가려져 방해를 받는 상태고 onResume은 현재 액티비티가 온전히 보이는 상태다.

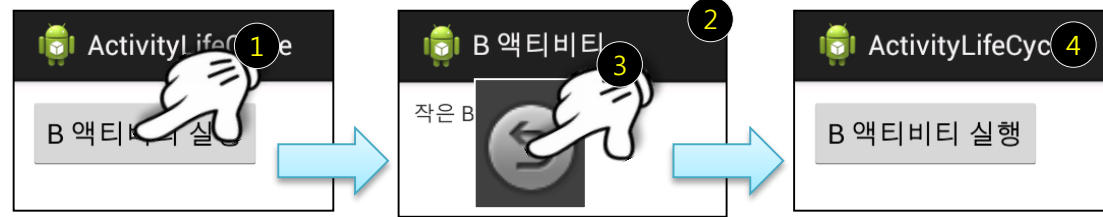
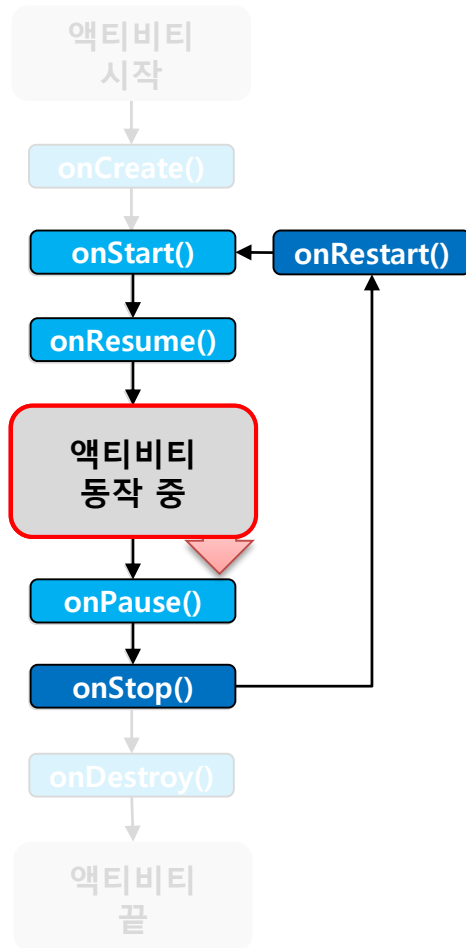
따라서 이 두 가지 함수는 화면이 조금이라도 방해 받지 않아야 할 때, 그 작업을 잠시 중단하거나 다시 시작하는 작업을 해주면 되겠다.

예를 들어 동영상을 보다가 화면이 가려지면 불편하다. 이때 onPause에서 동영상을 일시 정지하고 onResume에는 일시 정지된 동영상을 재생하면 되겠다.

그렇다고 onPause가 호출될때 무조건 그리는 것을 중단할 필요는 없다. 해당 액티비티가 완전히 가려진 것은 아니기 때문이다.



액티비티 상태 - 액티비티 정지와 재실행 상태

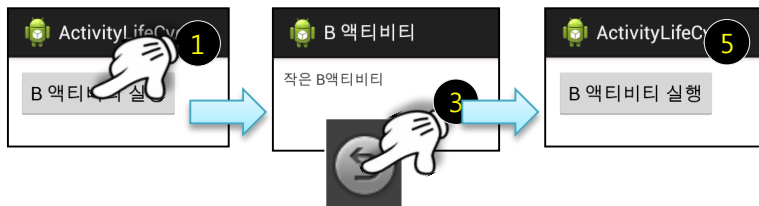


B 액티비티 뒤로
A 액티비티가
완전히 안 보인다.

액티비티 상태 - 액티비티 정지와 재실행 상태

androidManifest.xml

```
...  
<activity  
    android:name=".BActivity"  
    android:label="B 액티비티"  
    android:theme="@android:style/Theme.Dialog"/>  
  
</application>  
</manifest>
```



● 로그 출력 결과

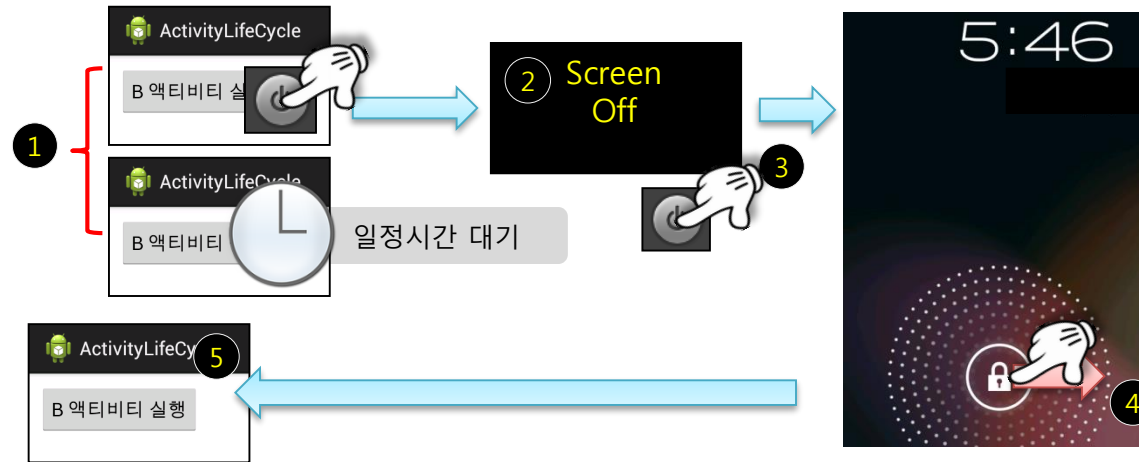
onCreate ()
onStart ()
onResume ()

2 onPause ()
onStop ()

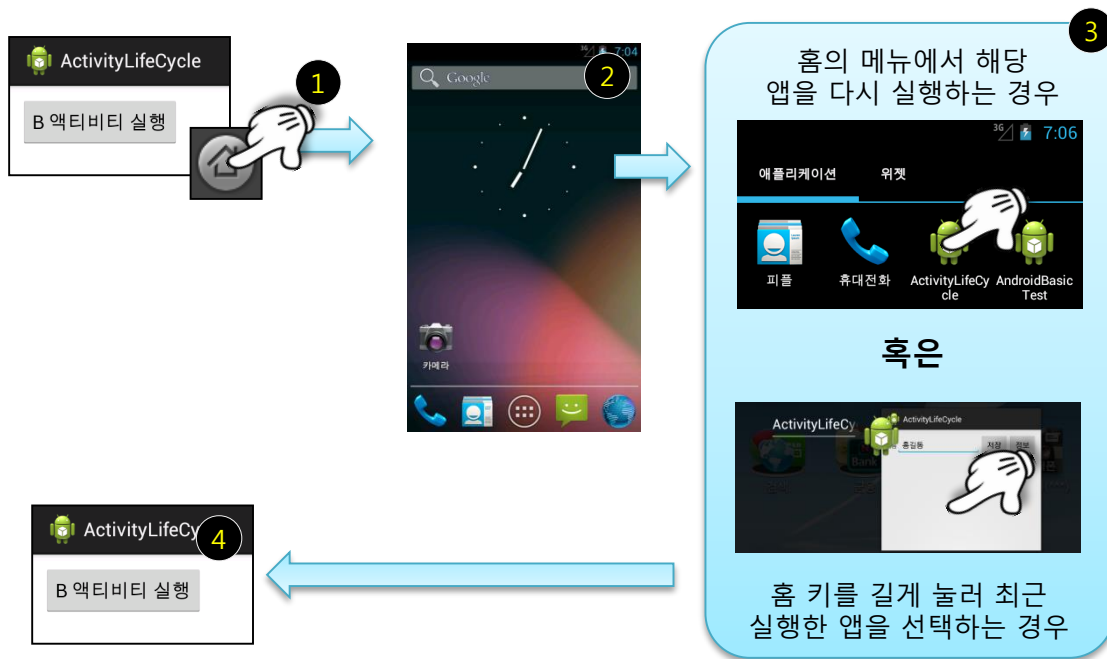
4 onRestart ()
onStart ()
onResume ()

액티비티 상태 - 액티비티 정지와 재실행 상태

■ 그 밖에 액티비티가 정지되는 상황들



액티비티 상태 - 액티비티 정지와 재실행 상태

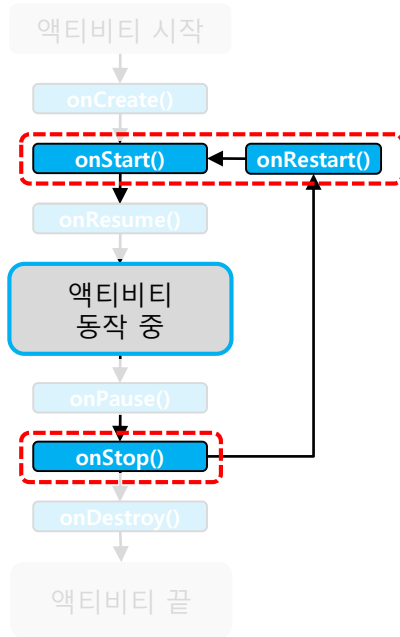


■ 화면이 가려진 액티비티가 정지되는 않는 경우



액티비티 상태 - 액티비티 정지와 재실행 상태

■ onStart와 짝을 이루는 onStop



- onStop은 현재 액티비티가 다른 액티비티로 부터 완전히 가려진 상태고 onStart는 현재 액티비티가 온전히 보이는 상태다.

따라서 onStop이 호출되면 더 이상 사용자는 액티비티를 볼 수 없기 때문에 그리는 작업은 중단해야 한다. onStart가 호출되면 중단된 작업을 재게하면 되겠다.

onRestart는 onStop 다음에 호출되는 유일한 경로다.

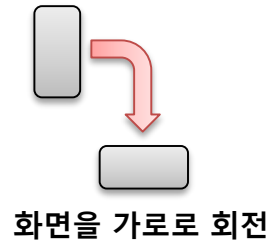
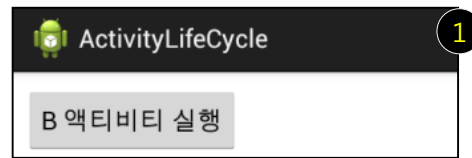
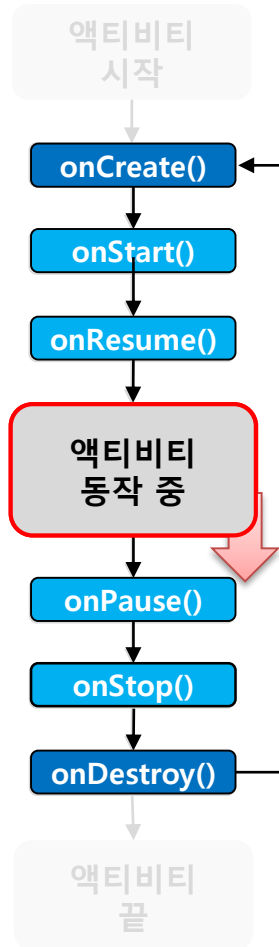
하지만 onStart는 onStop과 onCreate 두 곳에서 다음에 호출될 수 있어 구분이 어렵다. 이러한 구분이 중요한 경우에 onRestart를 활용할 수 있겠다.

참고로 onRestart는 많이 사용되지 않는다.

액티비티 상태 -

시스템 환경 변화에 의한 액티비티 강제 종료와 재실행 상태

5  
강제 종료/실행



액티비티 상태 -

시스템 환경 변화에 의한 액티비티 강제 종료와 재실행 상태

■ 왜 시스템 환경이 변하면 액티비티 생명주기가 다시 시작될까?

res

layout-land
activity_main.xml
layout-port
activity_main.xml

1

2

시스템 환경 변화로 레이아웃 리소스를 달리 적용해야 한다.



■ 예제로 직접 확인해보자.

세로 전용 화면

ActivityLifecycle

이름을 입력하세요.

저장

● 화면 구성

LinearLayout
EditText
Button

가로 전용 화면

ActivityLifecycle

이름을 입력하세요. 저장

● 화면 구성

LinearLayout
EditText
Button

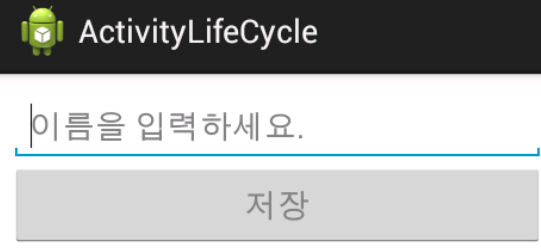
액티비티 상태 -

시스템 환경 변화에 의한 액티비티 강제 종료와 재실행 상태

res/layout-port/activity_main.xml

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

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="이름을 입력하세요." />
```



ActivityLifecycle

이름을 입력하세요.

저장

res/layout-land/activity_main.xml

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

    <EditText
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:hint="이름을 입력하세요." />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="저장" />

</LinearLayout>
```



ActivityLifecycle

이름을 입력하세요. 저장

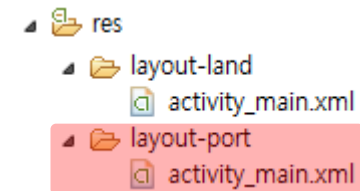
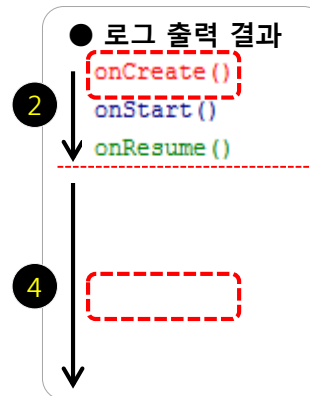
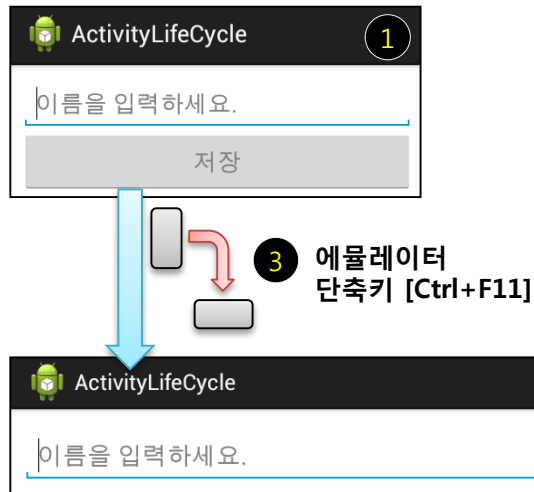
액티비티 상태 -

시스템 환경 변화에 의한 액티비티 강제 종료와 재실행 상태

src/MainActivity.java

```
public class MainActivity extends Activity
{
    ...
    protected void onCreate( Bundle savedInstanceState )
    {
        super.onCreate( savedInstanceState );

        setContentView( R.layout.activity_main );
    }
    ...
}
```



액티비티 상태 -

시스템 환경 변화에 의한 액티비티 강제 종료와 재실행 상태

■ 하지만 시스템 환경이 변해도 생명주기를 다시 시작할 필요가 없다면?

androidManifest.xml

```
...
<activity
    android:name="com.superdroid.test.activity.Lifecycle.MainActivity"
    android:label="@string/app_name"
    android:configChanges="orientation/screenSize">
...

```

src/MainActivity.java

```
public class MainActivity extends Activity
{
    ...

    @Override
    public void onConfigurationChanged( Configuration newConfig )
    {
        super.onConfigurationChanged( newConfig );

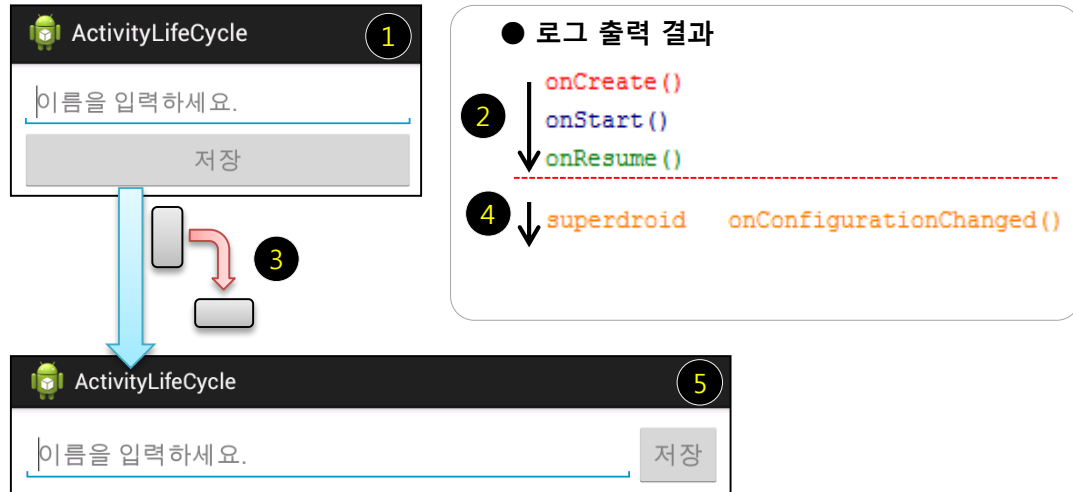
        // ① 로그를 남긴다.
        Log.w( "superdroid", "onConfigurationChanged()" );

        // ② 만일 시스템 환경이 변경되면 화면 레이아웃을 다시 설정한다.
        setContentView( R.layout.activity_main );
    }
}

```


액티비티 상태 -

시스템 환경 변화에 의한 액티비티 강제 종료와 재실행 상태



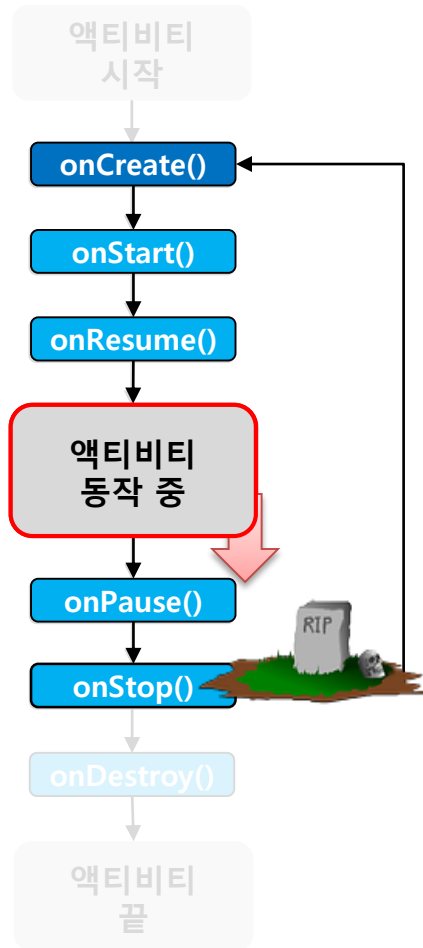


orientation과 screenSize 속성값

화면을 회전하는 경우 화면 방향만 바뀌는 것이 아니라 화면의 너비와 높이도 서로 바뀐다. 따라서 orientation 속성만 설정하고 화면을 회전하면 생명주기 함수들이 계속 호출된다. 따라서 반드시 화면 너비와 높이 변경을 감지하는 screenSize 속성값도 같이 추가해야 한다. 참고로 screenSize값은 안드로이드 API 13부터 추가되었다. 따라서 그 이전까지는 orientation값만 설정해도 문제가 없었다.

액티비티 상태 - 시스템에 의한 액티비티 강제 종료와 재실행 상태

5 강제 종료/실행



안드로이드 시스템은 앱 프로세스 상태를 포그라운드와 백그라운드 두 가지로 구분하고, **시스템 메모리가 부족해지면 백그라운드 상태의 앱 프로세스를 강제로 종료하여 자원을 확보한다.**

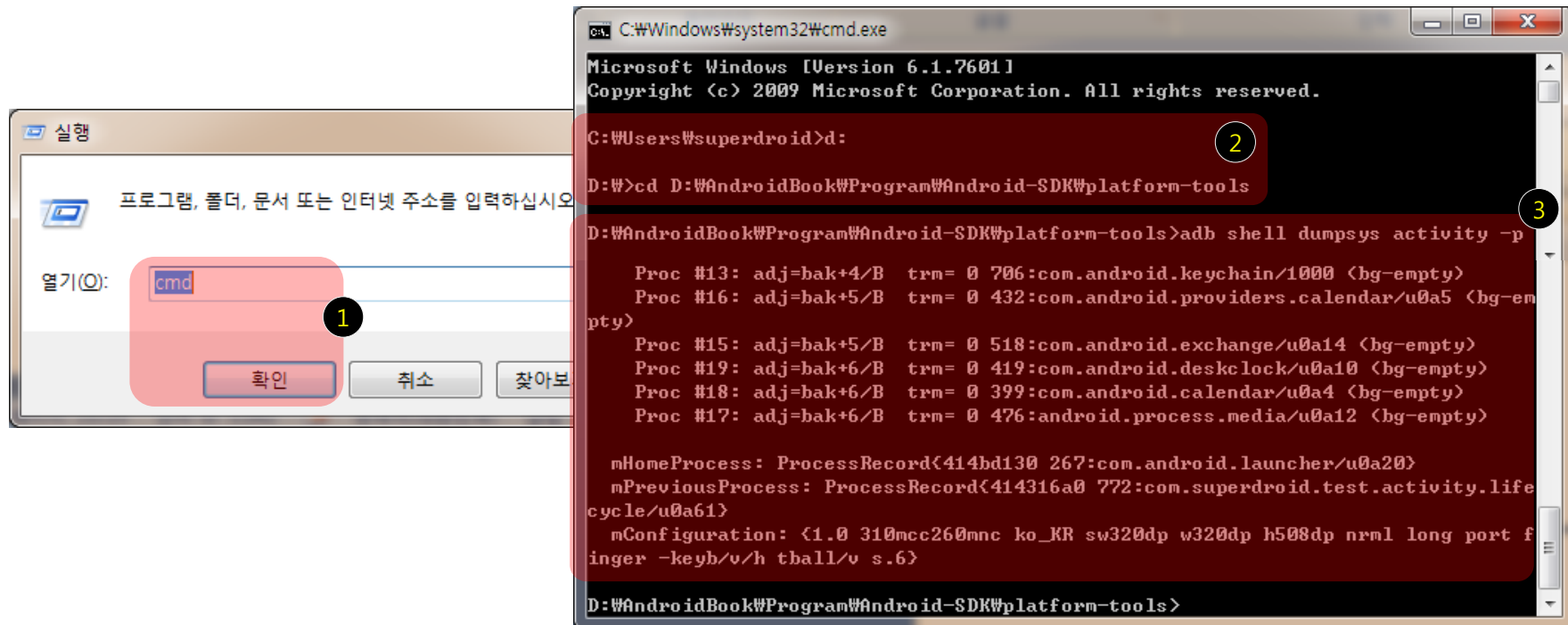
여기서 포그라운드는 앱의 액티비티가 화면에 보이는 상태고, 백그라운드는 다른 앱 액티비티에 완전히 가려져 보이지 않는 상태를 말한다.

액티비티 상태 - 시스템에 의한 액티비티 강제 종료와 재실행 상태

■ 포그라운드 앱과 백그라운드 앱의 상태를 직접 확인해보자.

통해 포그라운드와 백그라운드 앱 프로세스를 확인해보자. 앱 프로세스 상태를 확인하기 위해서는 ADBAndroid Debug Bridge 라는 툴을 이용해 단말기 내부로 접속해야 한다. ADB 툴은 다음의 경로에 존재한다.

- 안드로이드 SDK 폴더/platform-tools/adb.exe
- 현재 실행중인 앱 프로세스 상태보기 : `adb shell dumpsys activity -p` > *appProcessInfo.txt*
해당 파일로 결과를 저장

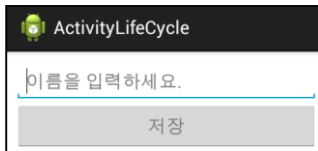


액티비티 상태 - 시스템에 의한 액티비티 강제 종료와 재실행 상태

appProcessInfo.txt

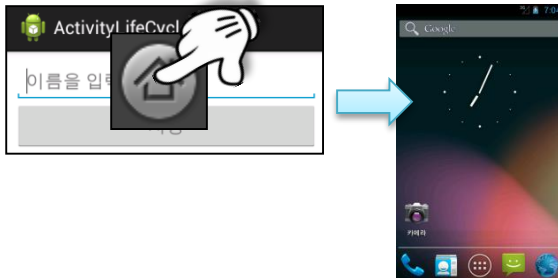
```
...  
ACTIVITY MANAGER RUNNING PROCESSES (dumpsys activity processes)  
Process LRU list (sorted by oom_adj):  
  PERS # 3: adj=sys /F trm= 0 149:system/1000 (fixed)  
  PERS #20: adj=pers /F trm= 0 246:com.android.phone/1001 (fixed)  
  PERS # 0: adj=pers /F trm= 0 218:com.android.systemui/u0a39 (fixed)  
  Proc # 5: adj=fore /FA trm= 0 267:com.android.launcher/u0a20 (top-activity)  
  Proc # 1: adj=vis /F trm= 0 233:com.android.inputmethod.latin/u0a19  
  Proc # 6: adj=prev /B trm= 0 772:com.superdroid.test.activity.lifecycle/u0a61 (previous)  
...
```

● 앱이 화면에 보이는 상태



Proc # 0: adj=fore /FA trm= 0 772:com.superdroid.test.activity.lifecycle/u0a61 (top-activity)

● 홈 키로 인해 앱이 화면에 보이지 않는 상태



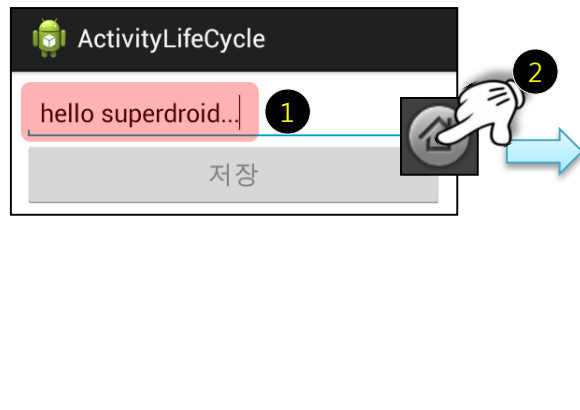
Proc # 5: adj=prev /B trm= 0 772:com.superdroid.test.activity.lifecycle/u0a61 (previous)

액티비티 상태 - 시스템에 의한 액티비티 강제 종료와 재실행 상태

■ 시스템이 메모리 부족 시 강제로 종료될 수 있는 백그라운드 상태는

매우 위험하고 불안한 상태가 아닌가?

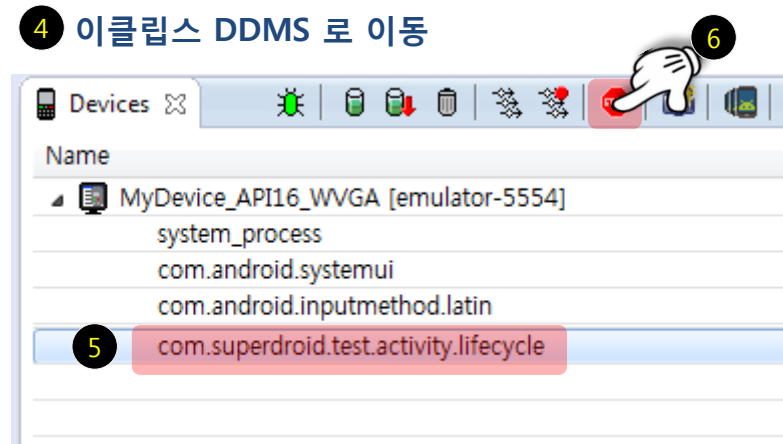
예를 들어 아주 긴 메모를 작성하고 있던 중 잠시 홈으로 이동한 사이 시스템이 강제로 백그라운드 앱 프로세스를 종료해버린다면, 애써 작성한 메모 내용이 모두 날아가 버릴 것이다. 잠시 그 상황을 예제 앱을 실행하여 확인해보자.



● 로그 출력 결과

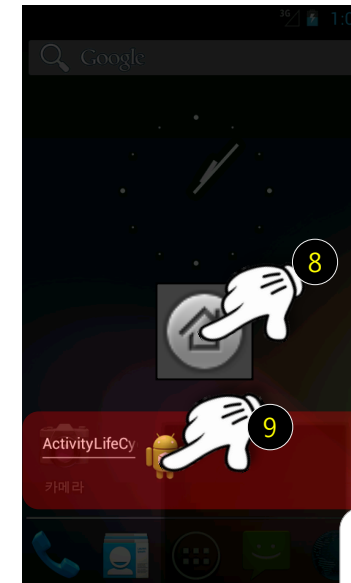
3
↓
`onCreate ()`
`onStart ()`
`onResume ()`
`onPause ()`
`onStop ()`

4 이클립스 DDMS 로 이동



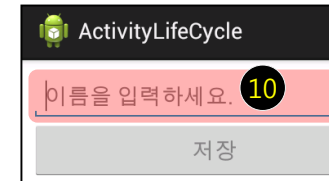
● 로그 출력 결과

7
↓
`onCreate ()`
`onStart ()`
`onResume ()`
`onPause ()`
`onStop ()`



● 로그 출력 결과

11
↓
`onCreate ()`
`onStart ()`
`onResume ()`
`onPause ()`
`onStop ()`
`onCreate ()`
`onStart ()`
`onResume ()`

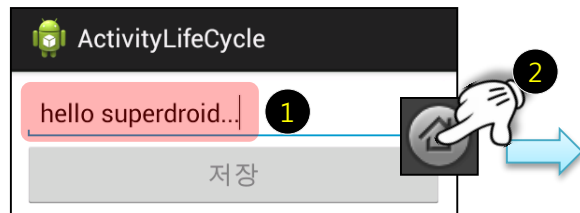


액티비티 상태 - 시스템에 의한 액티비티 강제 종료와 재실행 상태

■ 시스템이 메모리 부족 시 강제로 종료될 수 있는 백그라운드 상태는

매우 위험하고 불안한 상태가 아닌가?

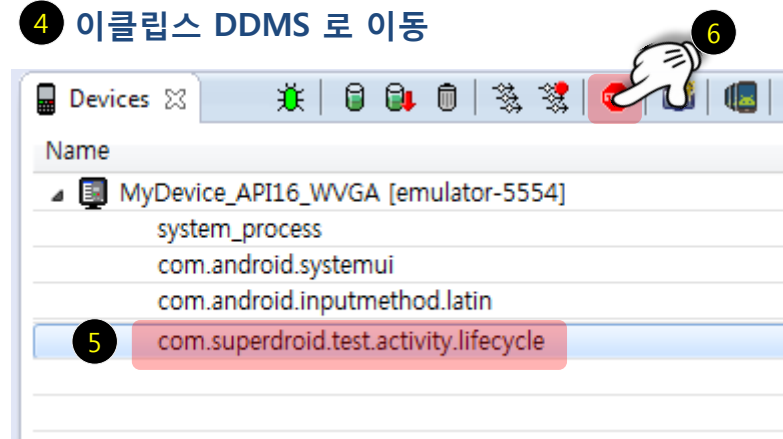
예를 들어 아주 긴 메모를 작성하고 있던 중 잠시 홈으로 이동한 사이 시스템이 강제로 백그라운드 앱 프로세스를 종료해버린다면, 애써 작성한 메모 내용이 모두 날아가 버릴 것이다. 잠시 그 상황을 예제 앱을 실행하여 확인해보자.



● 로그 출력 결과

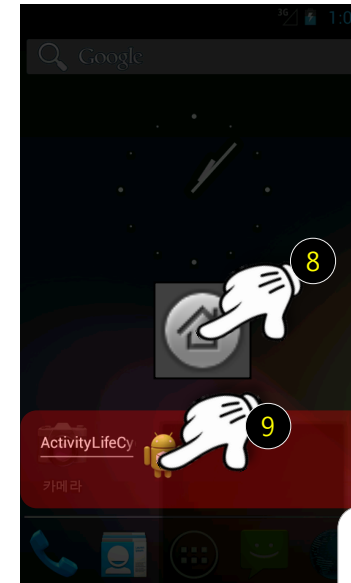
3
↓
onCreate ()
onStart ()
onResume ()
onPause ()
onStop ()

4 이클립스 DDMS 로 이동



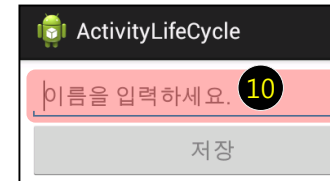
● 로그 출력 결과

7
↓
onCreate ()
onStart ()
onResume ()
onPause ()
onStop ()



● 로그 출력 결과

11
↓
onCreate ()
onStart ()
onResume ()
onPause ()
onStop ()
onCreate ()
onStart ()
onResume ()



■ 액티비티 상태 – 시스템에 의한 액티비티 강제 종료와 재실행 상태

■ 이거 원 불안해서... 어떻게 안드로이드 단말을 사용하라고...

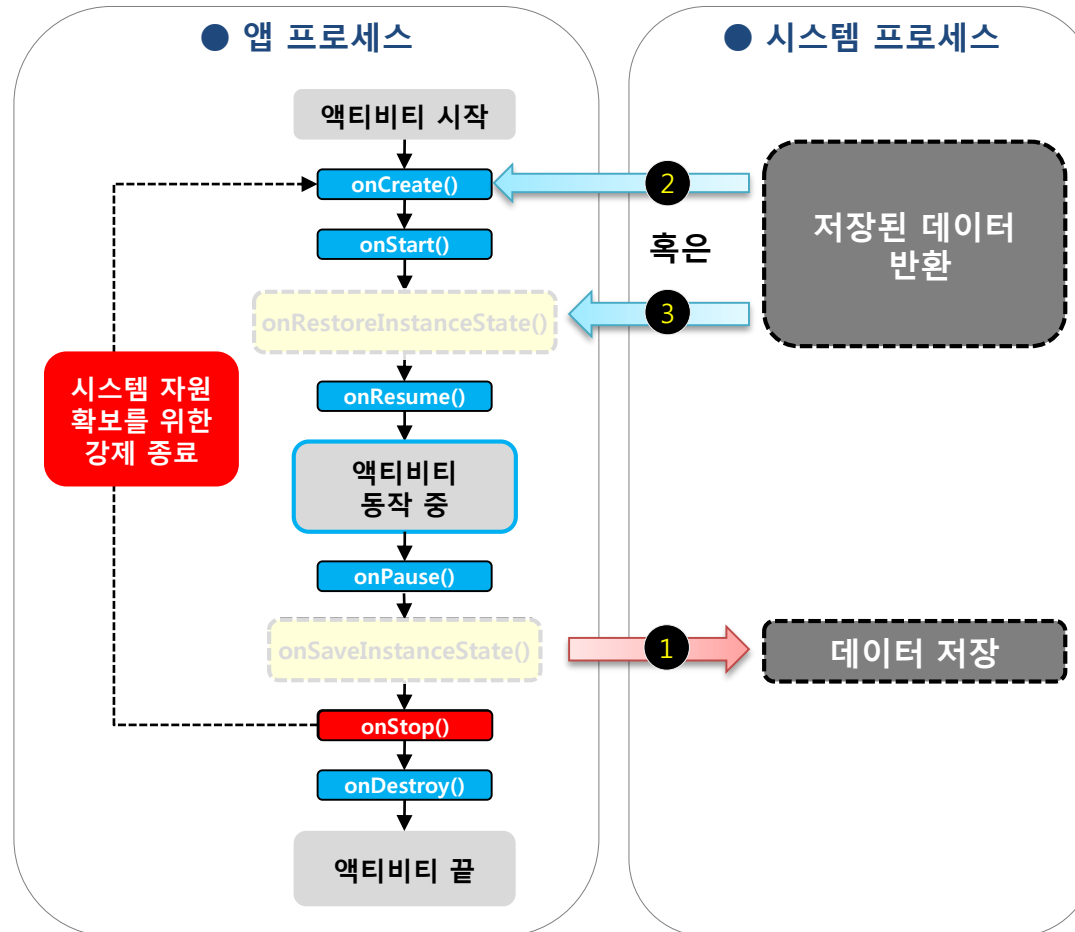
시스템에 의한 강제 종료는 사용자에게 매우 당황스러운 일이 아닐 수 없다. 안드로이드는 동시에 여러 앱을 실행하는 멀티태스킹Multitasking을 지원하고, 홈 키를 이용하여 다른 앱을 실행할 수 있다. 하지만 잠시 다른 앱을 사용하던 중 이전 앱이 종료되어 버린다면 불안해서 어떻게 안드로이드 단말을 사용할 수 있을까?

이를 위해 안드로이드는 강제 종료된 앱의 액티비티를 복원할 수 있는 방법을 제공하고 있다.

액티비티 데이터 복원

- 안드로이드에서는 앱 액티비티에서 사용된 데이터를 복원하기 위해 다른 프로세스로 데이터를 잠시 백업해두는 방법을 취한다.

여기서 다른 프로세스는 액티비티 매니저가 존재하는 시스템 프로세스를 말한다.



src/MainActivity.java

```
public class MainActivity extends Activity
```

```
{
```

```
    EditText mEditText = null;
```

```
    protected void onCreate( Bundle savedInstanceState ) {
```

```
        super.onCreate( savedInstanceState );
```

```
        setContentView( R.layout.activity_main );
```

```
        Log.e( "superdroid", "onCreate()" );
```

```
        mEditText = (EditText) findViewById( R.id.edit_text );
```

```
    }
```

```
    protected void onSaveInstanceState(Bundle outState) {
```

```
        Log.w( "superdroid", "onSaveInstanceState()" );
```

```
        String backupString = mEditText.getText().toString();
```

```
        outState.putString( "BACKUP_STRING", backupString );
```

```
        super.onSaveInstanceState( outState );
```

```
    }
```

```
    protected void onRestoreInstanceState( Bundle savedInstanceState ) {
```

```
        Log.w( "superdroid", "onRestoreInstanceState()" );
```

```
        if( savedInstanceState != null ) {
```

```
            mEditText.setText( savedInstanceState.getString( "BACKUP_STRING" ) );
```

```
        }
```

```
        super.onRestoreInstanceState( savedInstanceState );
```

```
    }
```

```
    ...
```

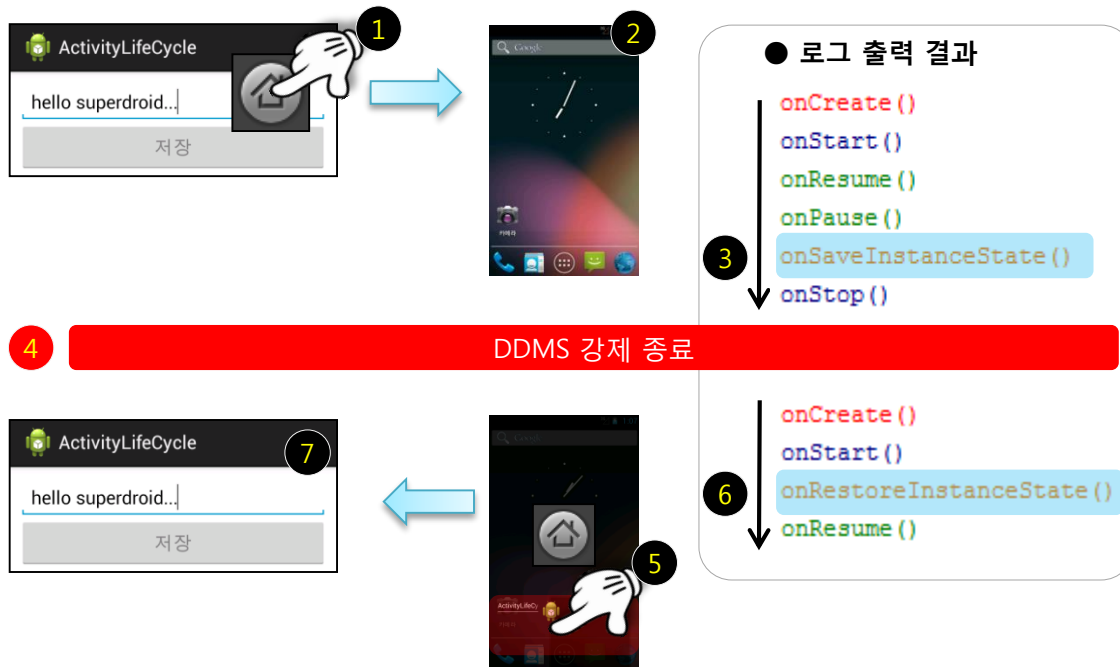


ActivityLifecycle

hello superdroid..|

저장

액티비티 데이터 복원



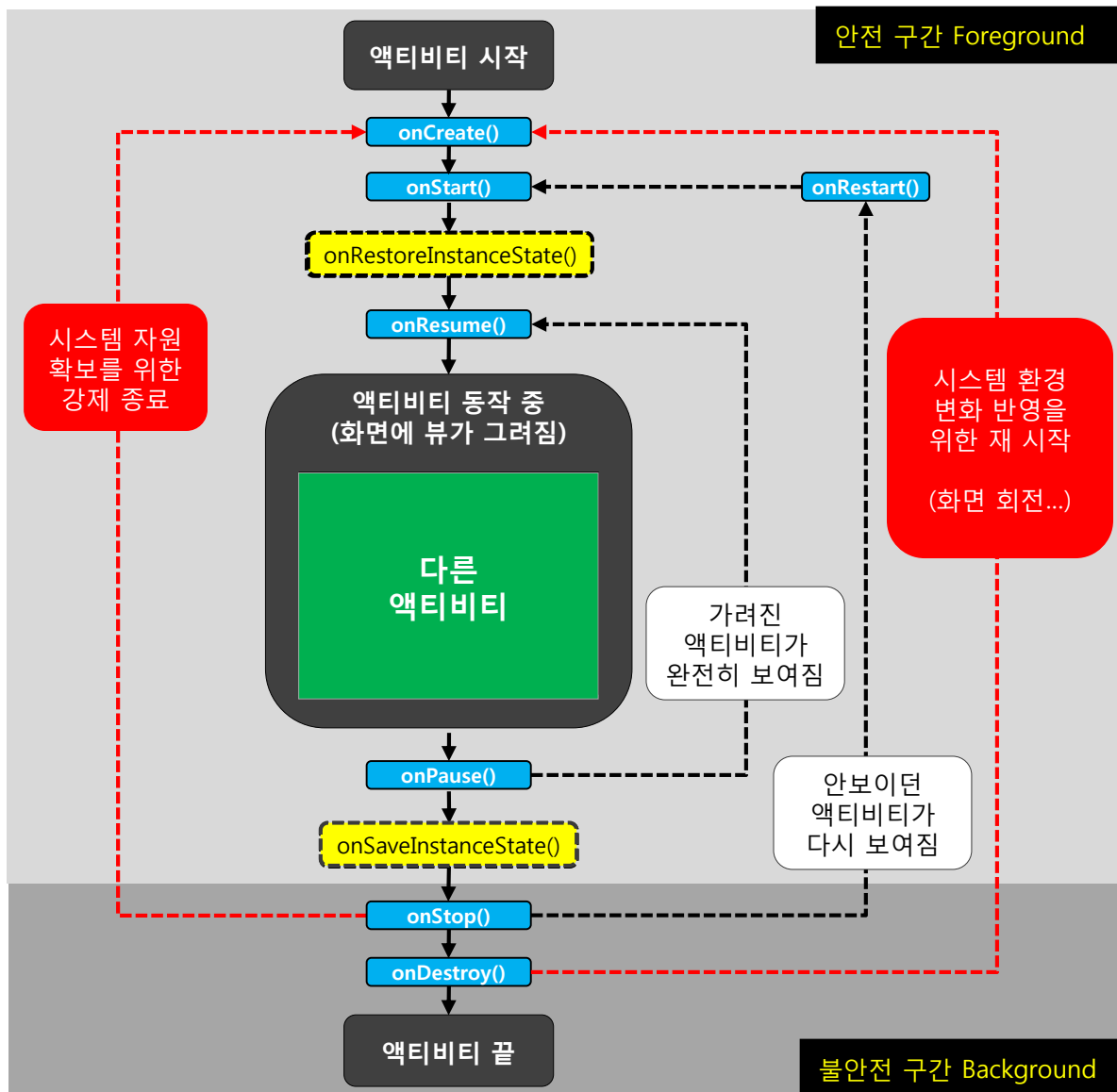
!!! 데이터 저장 크기는 절대 100KB를 넘지 말자.

시스템으로 데이터를 전달할 때 프로세스 통신을 사용하는데 이때 데이터를 전달하는 바인더 버퍼 사이즈의 제한이 있기 때문이다.

따라서 용량이 큰 데이터는 파일로 저장하고 저장된 파일 경로만 주고 받도록 하자.

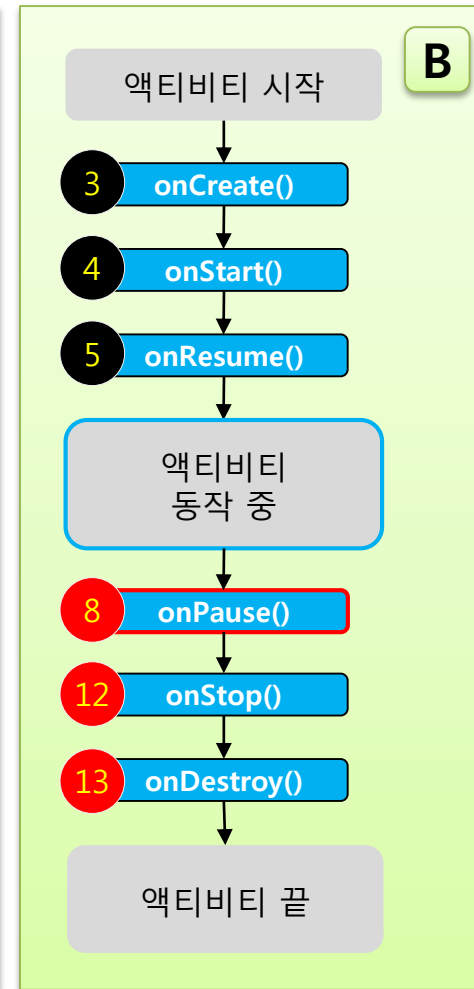
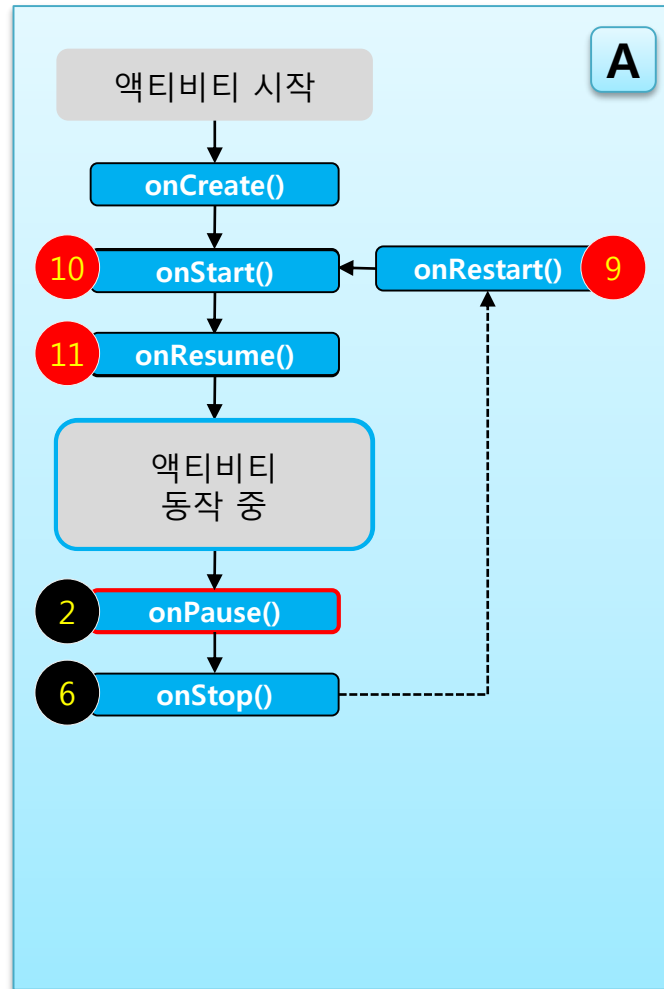
13장에서 상세히 설명한다.

한 눈에 보는 액티비티 생명주기



한 눈에 보는 액티비티 생명주기

■ A 액티비티에서 B 액티비티가 실행 및 종료 될 때 생명주기



실행중인 액티비티 onPause() → 실행될 액티비티 생명주기 → 실행 중이었던 액티비티 나머지 생명주기