

SECTION 01

액션바(ActionBar)

액션바(ActionBar)는 태블릿용 버전인 안드로이드 3.0(허니콤, API 레벨 11)부터 등장하여 안드로이드 4.0에서는 태블릿뿐만 아니라 폰 등의 기기에도 적용할 수 있게 변경되었다. 액션바라는 용어가 생소하기는 하지만 기존 타이틀바에 다양한 기능이 추가된 것이라고 생각하면 그리 어렵게 느껴지지 않을 것이다.

액션바를 통해 사용자가 현재 실행한 애플리케이션이 무엇인지 그리고 애플리케이션 내에서 사용자가 보고 있는 화면이 무엇인지를 알려줄 수 있으며, 탐색 메뉴를 통해 사용자가 애플리케이션을 쉽게 탐색할 수 있게 할 수 있다.

액션바의 주요 특징

- 애플리케이션 이름, 아이콘 그리고 애플리케이션 내의 사용자 위치를 알려주기 위한 영역을 제공
- 일관된 탐색 메뉴 제공
- 검색, 공유 등의 메뉴 영역을 제공

액션바는 다음과 같은 모양으로 구성된다.



그림 5-1

왼쪽에는 액티비티의 아이콘과 제목이 표시되며, 오른쪽에는 메뉴가 표시된다. 메뉴는 그림에서 보는 것처럼 아이콘만 보이게 할 수도 있고 아이콘과 제목이 함께 보이게 할 수도 있다. 또한 오른쪽 끝에는 더보기 메뉴 아이콘을 표시하여 추가 메뉴는 이 아이콘을 클릭했을 때만 보이게 할 수도 있다. 액션바에서 아이콘이 표시되는 메뉴들을 액션 아이템(Action Item)이라고 한다. 그리고 액션바 하단에는 실제 액티비티의 메인 영역이 표시된다.

액션바 만들기

참고 프로젝트 ActionBarDemoA1

간단히 액션바를 만들고 이 액션바가 WVGA(800*480)과 WXGA(1280*800)에서 어떻게 보이는지 확인해보도록 하겠다. 액션바를 만든다는 표현을 했지만 액션바는 타이틀바의 발전된 형태이기 때문에 별도로 만들 필요가 없다. 어떤 추가적인 작업을 하지 않아도 타이틀바가 보이는 것처럼 액션바도 보이게 된다. 그러므로 기본 프로젝트를 한 개 만들어서 실행하면 된다.

두 개의 에뮬레이터를 생성해서 기본 프로젝트를 실행한 결과는 다음과 같다. 첫 번째 그림은 WVGA(800*480) 해상도로 우리가 일반적으로 사용하는 폰의 해상도에서 실행한 결과이다. 그리고 두 번째 그림은 태블릿 해상도인 WXGA(1280*800) 에뮬레이터에서 실행한 결과다.



그림 5-2 WVGA(800*480)의 액션바

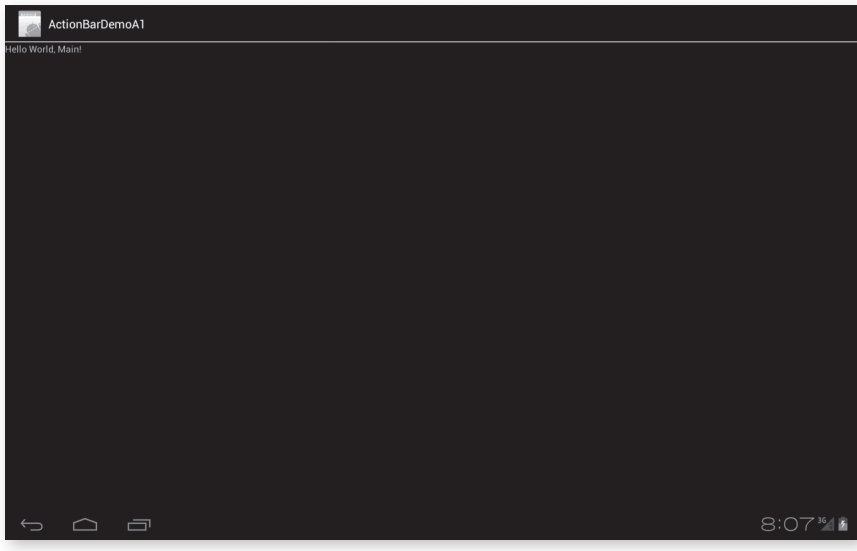


그림 5-3 WVGA(1200*800)의 액션바

WXGA(1280*800) 해상도의 액션바를 보면 오른쪽이 매우 허전함을 알 수 있다. 바로 이러한 이유로 인해 액션바에서는 액션바 오른쪽에 메뉴를 추가할 수 있게 하고 있다. 또한 액션바 중앙에 검색 뷰와 같은 별도의 뷰를 추가할 수 있게 하고 있다. 이러한 내용에 대해서는 차츰 알아보도록 하겠다.

기본 프로젝트에서 자동으로 생성된 레이아웃과 액티비티 코드는 다음과 같다. main.xml에서 @string/hello가 가리키는 문자열은 /res/values/strings.xml에 작성되어 있으며, 이 파일도 프로젝트 생성 시 기본으로 생성되는 파일이다.



CODE

코드 레이아웃 XML - /res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
</LinearLayout>
```

기본으로 생성된 액티비티다.



CODE

코드 Main.java

```
package com.androidside.actionbardemoa1;

import android.app.Activity;
import android.os.Bundle;

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

액션바에 메뉴 추가하기

참고 프로젝트 ActionBarDemoA2

안드로이드 3.0 이전에 사용하던 타이틀바에서는 단지 제목 문자열만 표시할 수 있었지만(이미지를 넣거나 레이아웃을 바꾸는 등의 수정이 가능하긴 했다), 3.0 이후부터는 타이틀바가 액션바로 변경되었고, 이 액션바에 아이콘과 제목 그리고 메뉴를 탑재할 수 있게 되었다.

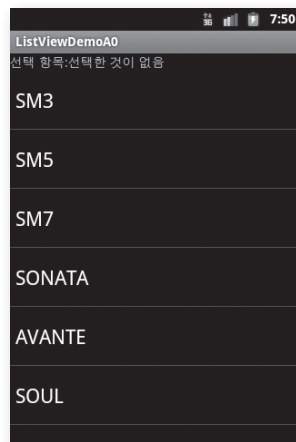


그림 5-4 안드로이드 2.3

안드로이드 3.0 이후 액션바는 기존 타이틀바와는 달리 매우 활용도가 높으며, 사용자가 애플리케이션을 좀 더 편하고 직관적으로 사용할 수 있게 도움을 준다. 그러므로 이 섹션에서는 액션바에 메뉴를 추가하고 이 메뉴를 어떻게 다룰 수 있는 지 살펴볼 것이다. 다음은 앞으로 우리가 만들 예제 프로젝트를 WVGA와 WXGA에 적용한 결과 화면이다.



그림 5-5 WVGA(800*480)의 액션바

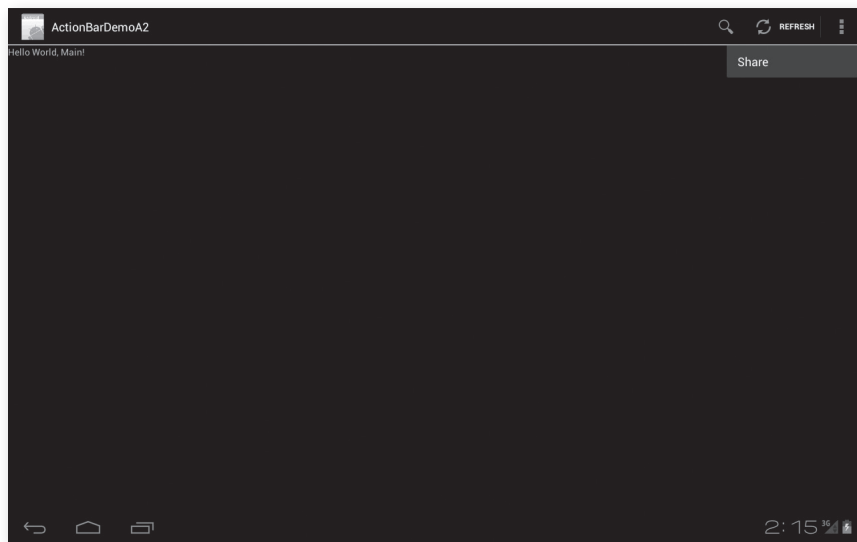



그림 5-6 WXGA(1280*800)의 액션바

두 화면의 차이점이 보이는가? 두 화면의 차이점을 이해하는 것은 폰과 태블릿 애플리케이션을 한번에 개발할 수 있는 안드로이드 4.0을 제대로 사용하는 첫 시작이 될 것이다. WVGA 화면에서는 상단 메뉴바에 모든 메뉴를 표시할 수가 없어 메뉴 버튼을 클릭했을 때 하단에 추가 메뉴를 보여준다. 이 방식은 안드로이드 3.0 이전의 방식이기도 하다. 물론 안드로이드 3.0 이전 버전에는 액션바가 없었으며, 메뉴는 모두 하단에 표시되었다. 두 번째 그림은 태블릿 크기인 WXGA에 적용한 메뉴를 보여준다. WXGA에서는 화면이 크기 때문에 오른쪽 상단에 많은 메뉴를 보여줄 수 있으며, 다 보여줄 수 없는 추가적인 메뉴는 더보기() 메뉴 버튼을 통해 추가적으로 볼 수 있다. 여기서 더보기 메뉴에 보이는 메뉴들은 개발자가 직접 정의할 수 있으며, 이 예제에서는 Share 메뉴는 더보기 메뉴로만 보이게 설정하고 있다.

또한 WXGA 화면에서는 REFRESH 문자열이 아이콘과 함께 보이고 있지만 WVGA 화면에서는 REFRESH 문자열이 보이지 않는다. 이는 화면 크기로 인해 안드로이드가 자동으로 보이지 않게 하고 있기 때문이다.

가장 먼저 이 예제에서 사용한 레이아웃은 기본 프로젝트 생성 시 자동으로 생성된 레이아웃이다.



CODE

레이아웃 XML - /res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />
</LinearLayout>
```

액션바에 설정된 메뉴는 안드로이드 3.0 이전에 작성하던 메뉴처럼 /res/menu/ 디렉터리 아래에 xml 파일로 작성하면 된다.



CODE

코드 메뉴 XML - /res/menu/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_refresh"
        android:title="@string/menu_refresh"
        android:icon="@drawable/ic_menu_refresh"
        android:orderInCategory="1"
        android:showAsAction="always|withText" />

    <item android:id="@+id/menu_search"
        android:title="@string/menu_search"
        android:icon="@android:drawable/ic_menu_search"
        android:orderInCategory="0"
        android:showAsAction="always" />

    <item android:id="@+id/menu_share"
        android:title="@string/menu_share"
        android:icon="@android:drawable/ic_menu_share"
        android:orderInCategory="1"
        android:showAsAction="never" />
</menu>
```

※ menu_search와 menu_share 아이템에서는 안드로이드에서 기본으로 제공하는 아이콘(@android:로 시작)을 사용하고 있지만, menu_refresh 아이템에서는 개발자가 직접 추가한 아이콘을 사용하고 있다. 이 예제를 실행하기 위해서는 @drawable/ic_menu_refresh가 가리키는 아이콘을 추가해야 하며, 필자는 /res/drawable-xhdpi에 아이콘을 추가했다.

메뉴의 첫 번째 아이템은 Refresh 버튼이며, 두 번째는 Search 버튼, 그리고 세 번째는 Share 버튼이다. 각각의 아이템에 있는 속성 중에서 관심있게 보아야 할 것은 바로 orderInCategory와 showAsAction이다. 먼저 orderInCategory 속성은 메뉴 순서를 지정하는 것이며, showAsAction은 해당 메뉴를 어떻게 보여줄지를 정하는 것이다. 이 속성에 always가 지정되면 액션바에 메뉴가 표시되며, never는 액션바에 보여주지 않고 더보기 메뉴로 보여지게 하는 속성이다. 이 속성으로 인해 위 두 개의 액션바 화면에서 Share 메뉴가 보이지는 위치가 달랐던 것이다.

android:showAsAction

현재 메뉴가 액션바의 액션 아이템으로 보여질 시점과 방식을 정의하는 속성.

이 속성은 자바 코드에서는 MenuItem 클래스의 setShowAsActionFlags() 메소드로 지정할 수 있으며, 이 메소드의 인자로 지정할 수 있는 상수 값은 다음 표에 정리하였다.

자바 코드 사용 예)

```
menuItem.setShowAsActionFlags(MenuItem.SHOW_AS_ACTION_IF_ROOM)
menuItem.setShowAsActionFlags(MenuItem.SHOW_AS_ACTION_IF_ROOM
    | MenuItem.SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW);
```



SUMMARY

액션뷰의 액션 아이템 showAsAction 속성

상수	설명
ifRoom	액션 아이템을 표시할 영역이 액션바에 있을 때만 표시한다. MenuItem 클래스에 정의된 상수 MenuItem.SHOW_AS_ACTION_IF_ROOM
withText	android:title로 정의된 문자열을 함께 표시한다. MenuItem 클래스에 정의된 상수 MenuItem.SHOW_AS_ACTION_WITH_TEXT
never	액션바에 액션 아이템을 표시하지 않는다. 이 속성의 기본 값이다. MenuItem 클래스에 정의된 상수 MenuItem.SHOW_AS_ACTION_NEVER
always	액션바에 액션 아이템을 항상 표시한다. 이 속성을 사용할 경우 액션 아이템이 중첩되는 문제가 발생할 수 있으므로 주의해서 사용해야 한다. MenuItem 클래스에 정의된 상수 MenuItem.SHOW_AS_ACTION_ALWAYS
collapseActionView	액션 아이템과 연관된 액션뷰(android:actionViewLayout에 선언한)의 아이콘만 보일 수 있게 한다. 안드로이드 4.0(API 레벨 14)에서 추가됨 MenuItem 클래스에 정의된 상수 MenuItem.SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW



METHOD SUMMARY

android.view.MenuItem 클래스의 메소드

MenuItem

setShowAsActionFlags(int actionEnum)

액션 아이템을 액션바에 어떻게 표시할지를 지정한다. actionEnum에는 매개변수 actionEnum에 나열한 상수를 지정할 수 있으며, SHOW_AS_ACTION_ALWAYS, SHOW_AS_ACTION_IF_ROOM, SHOW_AS_ACTION_NEVER는 함께 지정할 수 없다. 이 3개의 옵션과 함께 지정할 수 있는 것은 SHOW_AS_ACTION_WITH_TEXT와 SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW이다.

매개변수

actionEnum SHOW_AS_ACTION_ALWAYS, SHOW_AS_ACTION_IF_ROOM, SHOW_AS_ACTION_NEVER, SHOW_AS_ACTION_NEVER, SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW를 지정 가능

android:orderInCategory

여러 액션 아이템이 나열되는 순서를 정의하는 속성.

메뉴 XML을 작성했다면 이제 메뉴 XML에서 android:title="@string/menu_refresh"처럼 사용한 문자열을 작성해야 한다. 이 문자열은 /res/values/strings.xml 파일에 작성하면 된다.



CODE

코드 문자열 XML - /res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, Main!</string>
    <string name="app_name">ActionBarDemoA2</string>

    <string name="menu_refresh">Refresh</string>
    <string name="menu_search">Search</string>
    <string name="menu_share">Share</string>
</resources>
```

이제 액티비티 코드를 살펴보도록 하자.



CODE

Main.java

```

package com.androidside.actionbardemoa2;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.widget.Toast;

public class Main extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        getActionBar().setHomeButtonEnabled(true);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater menuInflater = getMenuInflater();
        menuInflater.inflate(R.menu.main, menu);

        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case android.R.id.home:
                Toast.makeText(this, "Home", Toast.LENGTH_SHORT).show();
                break;

            case R.id.menu_refresh:
                Toast.makeText(this, "Refresh", Toast.LENGTH_SHORT).show();
                break;

            case R.id.menu_search:
                Toast.makeText(this, "Search", Toast.LENGTH_SHORT).show();
                break;
        }
    }
}

```

```

        case R.id.menu_share:
            Toast.makeText(this, "Share", Toast.LENGTH_SHORT).show();
            break;
    }
    return super.onOptionsItemSelected(item);
}
}

```

코드가 다소 길기는 하지만 특별히 새로운 코드는 보이지 않는다. 액션바에 적용되는 메뉴가 기존에 사용하던 옵션 메뉴이기 때문이다. 그러므로 옵션 메뉴를 사용하는 것처럼 코드를 작성하면 된다.

getActionBar().setHomeButtonEnabled(true);

이 코드는 현재 액티비티의 액션바를 얻고(getActionBar()) 상단 왼쪽에 있는 아이콘을 클릭했을 때 애플리케이션 홈으로 이동할 수 있게 해주는 코드다. 그렇기 때문에 이 코드가 적용되면 onOptionsItemSelected() 메소드에서 왼쪽 상단 아이콘의 클릭을 감지할 수 있다. 클릭했을 때 넘어오는 아이디는 android.R.id.home 이다. 일반적으로 홈으로 이동하게 하며 원한다면 다른 처리를 추가할 수도 있다.

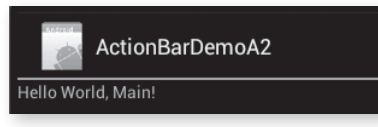


그림 5-7

메뉴에 대해서는 이미 “제4부 안드로이드 기초 - 메뉴”에서 살펴보았지만, 더 자세히 알고 싶다면 다음 링크를 참고하기 바란다.

URL <http://developer.android.com/guide/topics/resources/menu-resource.html>

액션바 옵션 살펴보기

액션바를 보이거나 숨기기

참고 프로젝트 ActionBarDemoB1

액션바는 현재 화면 위치와 관련된 메뉴를 보여줄 수 있어 매우 유용하지만, 경우에 따라서는 액션바로 인해 더 많은 정보를 보여줄 수 없는 경우도 있다. 그러므로 이런 경우에는 액션바를 숨겨서 화면을 좀 더 넓게 사용해야 할 필요가 있다.



그림 5-8 액션바를 숨긴 화면

액션바를 숨긴 후에도 원한다면 다시 보이게 할 수 있다. 숨길 때는 `ActionBar` 클래스의 `hide()`라는 메소드를 호출하면 되며, 보이게 할 때는 `show()` 메소드를 호출하면 된다.

현재 액티비티의 `ActionBar` 객체는 `getActionBar()` 메소드를 통해 얻을 수 있다. 그러므로 액션바를 숨기거나 보여주는 코드는 다음과 같다.

```
ActionBar actionBar = getActionBar();
actionBar.hide();
actionBar.show();
```

액션바에 제목, 부제목 설정하기

참고 프로젝트 ActionBarDemoB2

액션바의 왼쪽에는 아이콘과 현재 액티비티의 제목이 보여진다. 그런데 이 제목만으로는 해당 화면을 충분히 설명할 수 없는 경우가 있다. 이런 경우에는 다음처럼 제목 아래에 부제목을 추가할 수 있다.

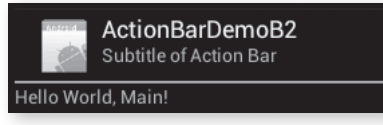


그림 5-9

ActionBarDemoB2는 제목이고, 이 제목 아래에 있는 Subtitle of Action Bar는 부제목이다. 이렇게 하기 위해서는 onCreate() 메소드에서 ActionBar 클래스의 setTitle() 메소드를 호출하면 된다.

```
getActionBar().setTitle("Subtitle of Action Bar");
```

기본적으로 제목은 /res/values/strings.xml에 선언되어 있는 app_name이 표시되지만, 원한다면 직접 제목을 변경할 수 있다. 제목 변경을 하고 싶을 때 사용하는 메소드는 setTitle()이다.

```
getActionBar().setTitle("Action Bar");
```

위 코드를 부제목을 설정하는 코드와 함께 호출하면 다음과 같은 화면이 출력된다.

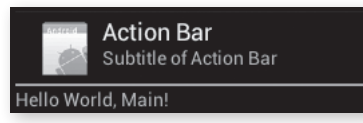


그림 5-10

추가적으로 제목과 부제목이 표시되는 영역을 보이지 않게 설정할 수도 있다.

```
getActionBar().setDisplayShowTitleEnabled(false);
```

다음은 제목과 부제목이 사라지고 아이콘만 보이는 모습이다.

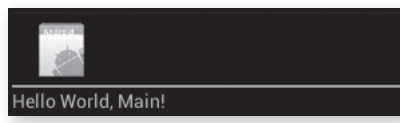


그림 5-11

액션바의 화면 출력 옵션 설정하기

참고 프로젝트 ActionBarDemoB3

액션바 왼쪽에는 아이콘과 제목, 부제목을 표시할 수 있다. 또한 아이콘 왼쪽에 < 표시를 하여 사용자가 아이콘을 클릭해서 이전 화면으로 이동할 수 있다는 것을 알려줄 수도 있다.

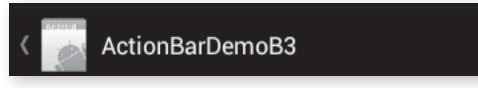


그림 5-12

이 섹션에서는 액션바 왼쪽에 아이콘, 제목 등을 표시할지 말지를 결정하는 옵션들에 대해서 살펴보고 하겠다. 가장 먼저 이를 지정하는 상수가 ActionBar 클래스에 정의되어 있다. 다음은 이를 정리한 것이다.



CONSTANTS

ActionBar 클래스에 지정된 액션바의 화면 출력 옵션

상수	설명
DISPLAY_USE_LOGO	<p>홈 아이콘 대신에 로고를 사용할지를 지정한다. 로고를 사용하고 싶다면 로고 이미지를 AndroidManifest.xml의 <application>이나 <activity>에 android:logo="@drawable/androidside"처럼 작성해야 한다.</p> <p>관련 메소드 setDisplayUseLogoEnabled(boolean useLogo)</p>
DISPLAY_SHOW_HOME	<p>홈 아이콘 표시 여부를 지정한다.</p> <p>관련 메소드 setDisplayShowHomeEnabled(boolean showHome)</p>
DISPLAY_HOME_AS_UP	<p>홈 아이콘 왼쪽에 < 표시를 하여 사용자에게 아이콘을 클릭 여부를 알려줄지를 지정한다.</p> <p>관련 메소드 setDisplayHomeAsUpEnabled(boolean showHomeAsUp)</p>
DISPLAY_SHOW_TITLE	<p>제목, 부제목을 표시 여부를 지정한다.</p> <p>관련 메소드 setDisplayShowTitleEnabled(boolean showTitle)</p>

상수	설명
DISPLAY_SHOW_CUSTOM	사용자 정의 뷰 표시 여부를 지정한다.
	관련 메소드 setCustomView(...) setDisplayShowCustomEnabled(boolean showCustom)

DISPLAY_로 시작하는 상수들은 액션바의 화면 옵션을 지정하는 것이며, 이를 활성화시키거나 비활성화시켜 액션바의 모양을 우리가 원하는 대로 변경할 수 있다. 이 상수들의 활성화 여부는 아래에서 살펴볼 `setDisplayOptions()` 메소드로 지정할 수 있으며, 이 메소드를 사용하지 않고 개별적으로 지정할 수도 있다. 이 메소드들을 “관련 메소드”라는 항목으로 위에 정리해줬으니 함께 살펴보기 바란다.

DISPLAY_로 시작하는 상수는 액션바의 일정 영역을 가리키고 있는데 이를 그림으로 정리해보면 다음과 같다. 그리고 `DISPLAY_SHOW_TITLE`을 비활성화시키면 액션바에는 제목이 보이지 않게 된다.

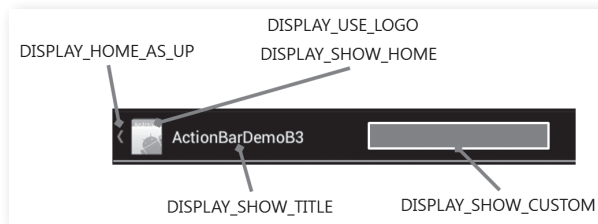


그림 5-13

DISPLAY_로 시작하는 상수를 가지고 활성화 여부를 지정하는 메소드는 다음과 같다. 두 개의 메소드가 준비되어 있는데 인자가 한 개인 메소드는 지정된 옵션만 활성화하며 인자가 두 개인 메소드는 여러 개의 옵션을 지정하여 활성화할 수 있다. 다음 표에 자세히 설명하였으니 꼭 읽어보기 바란다.

<div>API</div> METHOD SUMMARY	
android.app.ActionBar 클래스의 <code>setDisplayOptions</code> 메소드	
final Cursor	setDisplayOptions(int options) 화면 출력 옵션을 설정한다. options에 지정된 옵션만 활성화한다. 예를 들어, <code>setDisplayOptions(ActionBar.DISPLAY_SHOW_HOME)</code> 를 호출하면 <code>DISPLAY_SHOW_HOME</code> 만 활성화되며, <code>setDisplayOptions(ActionBar.DISPLAY_USE_LOGO ActionBar.DISPLAY_SHOW_HOME)</code> 를 호출하면 <code>DISPLAY_USE_LOGO</code> 와 <code>DISPLAY_SHOW_HOME</code> 가 활성화된다.

	매개변수 options ActionBar 클래스에 지정된 DISPLAY_로 시작하는 상수들의 조합
void	setDisplayOptions(int options, int mask) 화면 출력 옵션을 비트 마스크를 사용하여 설정한다. 예를 들어, setDisplayOptions(0, ActionBar.DISPLAY_SHOW_HOME)를 호출하면 DISPLAY_SHOW_HOME만 비활성화가 되며, setDisplayOptions(ActionBar.DISPLAY_SHOW_HOME, ActionBar.DISPLAY_SHOW_HOME DISPLAY_USE_LOGO)를 호출하면 mask에 선언된 두 개의 옵션 중에서 options에 선언된 DISPLAY_SHOW_HOME은 활성화 되고 그렇지 않은 DISPLAY_USE_LOGO는 비활성화된다.
	매개변수 options ActionBar 클래스에 지정된 DISPLAY_로 시작하는 상수들의 조합 mask 화면 출력 옵션의 비트 마스크



TIP & TECH

액션바에 아이콘 대신에 로고를 지정할 때

액션바에서 아이콘 대신에 로고를 표시하기 위해서는 setDisplayOptions() 메소드를 어떻게 호출해야 할까? 아마도 다음 코드처럼 작성하면 될 거라고 생각할 것이다.

```
setDisplayOptions(ActionBar.DISPLAY_USE_LOGO)
```

언뜻 보기에 이 코드는 로고를 사용하라고 지정하는 것이 맞다고 생각할 것이다. 하지만 막상 코드를 이렇게 작성하고 실행해보면 화면에 아무것도 나오지 않는다. 그래서 이를 해결하기 위해 AndroidManifest.xml에 logo가 제대로 지정되었는지, 이미지는 문제가 없는지 확인해볼 것이다. 하지만 이 부분도 문제가 없다는 것을 이내 깨닫고 한숨을 연거푸 내쉬는 개발자를 본 적이 있다. 무엇이 문제일까?

이 문제를 해결하기 위해서는 setDisplayOptions() 메소드는 지정된 옵션만 활성화시키는 것이고, 아이콘이나 로고를 표시하는 홈 영역은 DISPLAY_SHOW_HOME로 지정한다는 것을 유념해야 한다. 결국, 로고를 사용하고 싶다면 DISPLAY_USE_LOGO만을 활성화시킬 것이 아니라 DISPLAY_SHOW_HOME도 함께 활성화시켜야 하는 것이다. 다음 코드가 바로 우리가 원하는 코드다.

```
setDisplayOptions(ActionBar.DISPLAY_SHOW_HOME|ActionBar.DISPLAY_USE_LOGO);
```



그림 5-14 액션바에 로고를 표시한 화면

지금까지 살펴본 내용을 기반으로 간단한 예제를 작성해보겠다. 대표적인 화면 출력 옵션을 변경할 수 있도록 버튼을 추가하고 이 버튼을 통해 액션바의 왼쪽 영역을 변경해보겠다.



그림 5-15

버튼 4개를 추가해야 하므로 레이아웃은 다음처럼 구성하도록 한다.



CODE

레이아웃 XML - /res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/DISPLAY_HOME_AS_UP"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="DISPLAY_HOME_AS_UP | DISPLAY_SHOW_HOME" />

    <Button
        android:id="@+id/DISPLAY_SHOW_HOME"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="DISPLAY_SHOW_HOME" />

    <Button
        android:id="@+id/DISPLAY_USE_LOGO"
        android:layout_width="fill_parent"
```

```

        android:layout_height="wrap_content"
        android:text="DISPLAY_USE_LOGO | DISPLAY_SHOW_HOME" />

<Button
    android:id="@+id/DISPLAY_SHOW_TITLE"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="DISPLAY_SHOW_TITLE" />
</LinearLayout>

```

레이아웃을 작성했으니 이제 실제 코드를 작성해야 한다. 먼저 버튼을 클릭했을 때 반응할 수 있도록 onCreate() 메소드에 리스너를 설정한다. 그리고 onClick() 메소드에서 사용자가 클릭한 버튼의 종류에 따라 화면 출력을 적당히 변경하도록 한다.



CODE
Main.java

```

package com.androidside.actionbardemob3;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;

public class Main extends Activity implements View.OnClickListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        findViewById(R.id.DISPLAY_HOME_AS_UP).setOnClickListener(this);
        findViewById(R.id.DISPLAY_SHOW_HOME).setOnClickListener(this);
        findViewById(R.id.DISPLAY_USE_LOGO).setOnClickListener(this);
        findViewById(R.id.DISPLAY_SHOW_TITLE).setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.DISPLAY_HOME_AS_UP :
                getActionBar().setDisplayOptions(
                    ActionBar.DISPLAY_HOME_AS_UP|ActionBar.DISPLAY_SHOW_HOME);
                break;
            case R.id.DISPLAY_SHOW_HOME :
                getActionBar().setDisplayOptions(ActionBar.DISPLAY_SHOW_HOME);

```

```

        break;
    case R.id.DISPLAY_USE_LOGO :
        getActionBar().setDisplayOptions(
            ActionBar.DISPLAY_USE_LOGO|ActionBar.DISPLAY_SHOW_HOME);
        break;
    case R.id.DISPLAY_SHOW_TITLE :
        getActionBar().setDisplayOptions(ActionBar.DISPLAY_SHOW_TITLE);
        break;
    }

}

}

```

여기서 주의깊게 보아야 할 것은 `onClick()` 메소드에 있는 `setDisplayOptions()` 메소드다. 이 코드에서 사용한 메소드는 한 개의 인자만 받는 메소드이며, 이 메소드는 지정된 옵션만을 활성화시키고 나머지 화면 옵션은 모두 비활성화시킨다. 그러므로 `setDisplayOptions(ActionBar.DISPLAY_HOME_AS_UP)`처럼 코드를 작성하면 화면에 아무것도 보이지 않는다. 왜냐하면 `DISPLAY_HOME_AS_UP` 옵션은 홈 아이콘이 있을 경우에만 적용되는 옵션이기 때문이다. 그래서 두 개의 옵션이 모두 적용될 수 있도록 `ActionBar.DISPLAY_HOME_AS_UP|ActionBar.DISPLAY_SHOW_HOME`처럼 지정해야 한다.

마지막으로 `AndroidManifest.xml`에 로고 이미지를 추가한다. 로고를 사용하지 않는다면 몰라도 이 예제에서는 로고 출력을 하고 있으므로 `android:logo`에 이미지를 지정해야 한다. `android:logo`는 `<application>`이나 `<activity>`에 지정할 수 있다. 이 예제에서는 “안드로이드사이드” 로고를 지정했다. 필자는 이 예제에서 “안드로이드사이드” 로고를 `/res/drawable-hdpi`에 추가했다.



CODE

AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.androidside.actionbardemob3"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="15" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".Main"

```

```

        android:label="@string/app_name"
        android:logo="@drawable/androidside" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>

```

액션뷰 추가하기

액션바에 사용자 정의 뷰 추가하기

참고 프로젝트 ActionBarDemoC1

액션바에는 액션 아이템이라 불리는 메뉴를 추가해서 사용자의 편의를 제공할 수 있다. 하지만 메뉴는 단지 메뉴일 뿐이므로 버튼 등의 다른 뷰를 액션바에 추가해야 하는 경우도 있을 수 있다. 그래서 이 섹션에서는 액션바에 우리가 원하는 뷰를 어떻게 추가할 수 있는지를 살펴보도록 하겠다.

액션바를 생성하고, 액션바에 가장 대표적 뷰인 버튼을 추가하고, 버튼을 클릭했을 때 Toast로 메시지를 화면에 보이게 하겠다.

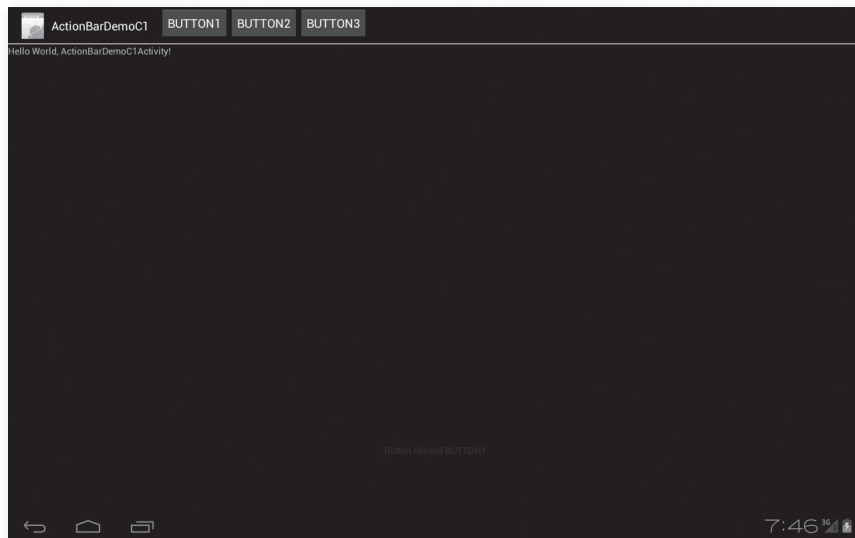


그림 5-16

이렇게 작성하기 위해서는 가장 먼저 버튼을 배치할 레이아웃을 작성해야 하며, 그 다음에 액션바에 사용자 정의 뷰를 사용하겠다고 선언해야 한다. 그리고 버튼을 클릭했을 때 메시지를 보여줄 수 있도록 코드를 작성해야 한다. 절차를 정리하면 다음과 같다.

- (1) 사용자 정의 뷰(버튼 레이아웃) 작성 - /res/layout/mybutton.xml
- (2) 사용자 정의 뷰(버튼 레이아웃) 액션바에 배치 - setContentView() 메소드
- (3) 사용자 정의 뷰(버튼 레이아웃) 활성화 - setDisplayShowCustomEnabled() 메소드
- (4) Toast 메시지 코드 작성 - Toast 클래스

가장 먼저 main.xml을 생성해야 하는데, 이 예제에서는 특별히 추가할 것이 없으므로 기본 생성 파일을 그대로 사용하겠다. 그러므로 우리가 작성해야 할 레이아웃은 버튼을 배치할 레이아웃 단 한 개뿐이다.



CODE

레이아웃 XML - /res/layout/mybutton.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <Button
        android:id="@+id/mybutton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON1" />

    <Button
        android:id="@+id/mybutton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON2" />

    <Button
        android:id="@+id/mybutton3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON3" />

</LinearLayout>
```

버튼 레이아웃을 작성했으니 이제 액션뷰에 이 버튼 레이아웃을 설정하고 활성화하면 된다.



CODE

Main.java

```
package com.androidside.actionbardemoc1;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class Main extends Activity implements View.OnClickListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        View myButtonLayout = getLayoutInflater().inflate(R.layout.mybutton, null);

        myButtonLayout.findViewById(R.id.mybutton1).setOnClickListener(this);
        myButtonLayout.findViewById(R.id.mybutton2).setOnClickListener(this);
        myButtonLayout.findViewById(R.id.mybutton3).setOnClickListener(this);

        ActionBar actionBar = getActionBar();
        actionBar.setCustomView(myButtonLayout);
        actionBar.setDisplayShowCustomEnabled(true);
    }

    @Override
    public void onClick(View v) {
        Toast.makeText(this, "Button clicked " + ((Button)v).getText(),
            Toast.LENGTH_SHORT).show();
    }
}
```

View myButtonLayout = getLayoutInflater().inflate(R.layout.mybutton, null);

작성한 버튼 레이아웃을 액티비티에서 사용하기 위해서는 인플레이션 작업이 필요하다.

myButtonLayout.findViewById(R.id.mybutton1).setOnClickListener(this);

인플레이션한 mybutton.xml의 버튼들을 찾아서 클릭 리스너를 등록한다.

```
ActionBar actionBar = getActionBar();
actionBar.setCustomView(myButtonLayout);
```

액션바에 버튼 레이아웃을 설정한다.

```
actionBar.setDisplayShowCustomEnabled(true);
```

버튼 레이아웃을 활성화한다.

기본적으로 액션바에 뷰를 추가하는 것은 매우 단순한 일이므로 실제 애플리케이션을 작성할 때 중요하게 생각할 것은 액션바에 직접 정의한 뷰의 기능을 정의하는 것이다. 이 기능 정의는 대부분 클릭 이벤트에 의해 처리될 것이므로 `onClick()` 메소드를 잘 정의하면 된다.

액션바에 사용자 검색뷰 추가하기

참고 프로젝트 ActionBarDemoC2

이전 섹션에서 사용자가 직접 작성한 뷰를 액션바에 추가했다. 이와 다르게 이미 제공되는 액션뷰를 추가할 수도 있다. 이 중에서 가장 활용도가 높은 것이 검색뷰다. 이 검색뷰는 주로 액션뷰로 사용하기는 하지만 위젯의 일종이기 때문에 다른 곳에 사용해도 된다.



그림 5-17

이 그림에서 보이는 상단 오른쪽에 있는 검색뷰는 검색 입력을 바로 할 수 있게 검색 입력 창을 보일 수도 있고, 돋보기 아이콘만 먼저 보이고 사용자가 클릭했을 때 검색 입력 창을 보이게 할 수도 있다. 이 섹션에서는 이러한 검색뷰를 액션바에 추가하고 이와 관련된 속성들을 살펴보도록 하겠다.

액션뷰는 액션바에서는 메뉴로 동작하기 때문에 작업 순서는 다음과 같다.

- (1) 메뉴 파일에 검색뷰 선언하기 – `/res/menu/main.xml`
- (2) 액티비티에서 검색뷰 속성 설정하기

일단 기본 프로젝트를 생성한 후에 수정해야 할 파일은 딱 두 가지다. `/res/menu` 디렉터리에 `main.xml`을 생성하고, 이 파일에 다음처럼 검색뷰를 메뉴로 선언한다. 이렇게 선언한 메뉴는 액션바에서 액션뷰의 형태로 동작하게 된다.



CODE

메뉴 XML - /res/menu/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_search"
        android:title="@string/menu_search"
        android:icon="@android:drawable/ic_menu_search"
        android:actionViewClass="android.widget.SearchView" />
</menu>
```

액션 아이템을 액션뷰로 선언하는 방식은 두 가지가 있다. 위에서 선언한 `actionViewClass` 속성에 액션뷰로 사용할 클래스를 직접 지정하는 방식과 `actionLayout` 속성에 액션뷰로 사용할 레이아웃을 지정하는 방식이 있다.

(1) actionViewClass 속성 사용

`actionViewClass` 속성을 사용하는 방식은 매우 단순하다. 이 속성에 액션뷰로 사용할 클래스를 다음처럼 패키지명을 포함하여 지정하기만 하면 된다. 물론 지정하는 위젯이 액션뷰로 사용할 수 있도록 미리 준비되어 있어야 한다. 이러한 뷰들이 많이 준비되어 있으면 좋겠지만 아쉽게도 현재는 검색뷰(SearchView)밖에 없다.

```
android:actionViewClass="android.widget.SearchView"
```

(2) actionLayout 속성 사용

`actionLayout` 속성에 레이아웃 위치를 지정하고 이 레이아웃에 검색뷰를 지정하는 방식이다. 그러므로 가장 먼저 /res/layout 디렉터리에 `mysearch.xml`를 생성해서 `SearchView` 위젯을 선언해야 한다.



CODE

레이아웃 XML - /res/layout/mysearch.xml

```
<?xml version="1.0" encoding="utf-8"?>
<SearchView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="match_parent" />
```

그리고 메뉴 레이아웃의 `actionLayout`에 방금 작성한 레이아웃을 지정하면 된다.



CODE

메뉴 XML — /res/menu/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_search"
          android:title="@string/menu_search"
          android:icon="@android:drawable/ic_menu_search"
          android:actionLayout="@layout/mysearchview" />
</menu>
```

두 번째 방식은 별도의 레이아웃을 생성해야 하는 번거로움이 있지만 액션뷰로 사용할 레이아웃을 따로 관리할 수 있다는 장점도 있다. 그러나 필자는 별도의 파일을 생성하지 않아도 되는 첫 번째 방식을 선호한다.

액션 아이템을 선언했다면 이제 액티비티를 작성해야 한다. 먼저 액티비티 전체 코드를 살펴보겠다.



CODE

Main.java

```
package com.androidside.actionbardemoc2;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.SearchView;
import android.widget.SearchView.OnQueryTextListener;

public class Main extends Activity implements OnQueryTextListener {
    private final boolean IS_ICON_ITEM = true;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        MenuItem searchMenu = menu.findItem(R.id.menu_search);

        SearchView searchView = (SearchView) searchMenu.getActionView();
```

```

searchView.setOnQueryTextListener(this);
searchView.setSubmitButtonEnabled(true);
searchView.setQueryHint("input query hint");

if (IS_ICON_ITEM) {
    searchMenu.setShowAsActionFlags(MenuItem.SHOW_AS_ACTION_IF_ROOM
        | MenuItem.SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW);
} else {
    searchMenu.setShowAsActionFlags(MenuItem.SHOW_AS_ACTION_IF_ROOM);
    searchView.setIconifiedByDefault(false);
}

return true;
}

@Override
public boolean onQueryTextChange(String newText) {
    return false;
}

@Override
public boolean onQueryTextSubmit(String query) {
    return false;
}
}

```

작성한 액티비티는 검색뷰를 보여주지만 하므로 검색어를 입력하고 검색 버튼을 클릭해도 어떤 반응도 하지 않는다. Main 클래스 상단에 선언되어 있는 IS_ICON_ITEM 상수는 검색뷰를 아이콘으로 표시할 지, 아니면 검색창으로 표시할지를 지정하는 불린 값이다. 이 값을 변경해서 실행한 화면은 다음과 같다.

(1) IS_ICON_ITEM = true

IS_ICON_ITEM을 true로 지정한다는 것은 아이콘으로만 표시하겠다는 것이고 이 값에 의해 다음과 같은 코드가 실행된다.

```

searchMenu.setShowAsActionFlags(MenuItem.SHOW_AS_ACTION_IF_ROOM
    | MenuItem.SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW);

```

setShowAsActionFlags() 메소드는 액션 아이템의 형태 및 표시 방식을 지정하는 메소드이며, 여기서는 액션뷰가 이 메소드에 의해 설정된다. 이에 대한 설명은 이전에 살펴본 “액션바에 메뉴 추가하기”를 참고하기 바란다.

IS_ICON_ITEM이 true일 경우에는 다음처럼 오른쪽에 검색 아이콘이 표시된다. 그리고 검색 버튼을 클릭하면 검색창이 보이게 된다.



그림 5-18

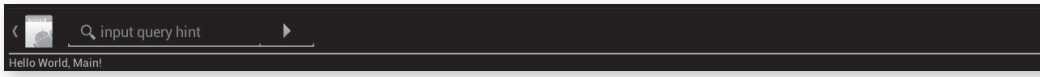


그림 5-19

왼쪽 아이콘을 클릭하면 검색 아이콘만 보이는 화면으로 돌아간다.

(2) IS_ICON_ITEM = false;

IS_ICON_ITEM을 false로 지정한다는 것은 액션바에 검색 아이콘이 아닌 검색창을 표시하겠다는 의미다. 그러므로 이 값에 의해 다음 코드가 실행된다.

```
searchMenu.setShowAsActionFlags(MenuItem.SHOW_AS_ACTION_IF_ROOM);
searchView.setIconifiedByDefault(false);
```

두 개의 코드 중에서 첫 번째 코드는 이전에 이미 살펴본 내용이고, 두 번째 코드의 setIconifiedByDefault() 메소드는 아이콘 표시 여부를 설정하는 메소드다. 그러므로 이 메소드에 false로 지정하여 검색 아이콘만 표시되게 하였다.



그림 5-20

```
getMenuInflater().inflate(R.menu.main, menu);
```

```
MenuItem searchMenu = menu.findItem(R.id.menu_search);
```

이 코드는 /res/menu/main.xml에 있는 메뉴 아이템을 자바 코드에서 사용하기 위한 코드다. 첫 번째 코드의 inflate() 메소드는 xml에 선언된 메뉴를 자바 코드에서 사용할 수 있도록 해주며, findItem() 메소드는 xml에 선언된 메뉴 아이템 중에서 원하는 메뉴 아이템을 찾아주는 메소드다.

```
SearchView searchView = (SearchView) searchMenu.getActionView();
```

getActionView() 메소드는 메뉴 아이템에 선언된 검색뷰의 객체를 생성해준다.

지금까지 액션바에 사용자 검색뷰를 추가하는 방법에 대해서 살펴보았다. 어렵지 않은 내용이지만 기존 개발자들에게는 생소한 내용이므로 관련된 메소드와 속성들에 대해서 차분히 살펴볼 필요가 있다.



TIP & TECH

액션바의 `iconifiedByDefault` 속성과 `showAsAction` 속성의 관계

`android:iconifiedByDefault` 속성은 액션 아이템을 아이콘으로 표시하는 속성이며, `android:showAsAction` 속성은 액션 아이템의 표시 여부와 표시 형태를 지정하는 속성이다. 그런데 이 두 개의 속성이 서로 밀접한 관계를 형성하고 있기 때문에 속성을 제대로 이해하고 사용할 필요가 있다. 예를 들어, `showAsAction`을 지정하지 않고 `iconifiedByDefault`만 `true`로 지정한다면 어떻게 될까? 일반적으로 아이콘으로 표시가 될 것이라고 생각하겠지만 액션바에 아이콘이 표시되지 않는다. 왜냐하면 `showAsAction`의 디폴트 속성은 액션바에 어떤 것도 표시하지 않는 것이기 때문이다. 그러므로 이 둘의 연관 관계를 이해하고 코드를 작성해야 한다. 물론 이에 대한 설명은 이 책에서 모두 다뤘으므로 액션바의 메뉴 부분과 방금 살펴본 코드들을 다시 한 번 살펴보면 될 것이다.

이 속성들과 관련된 자바 메소드 `setShowAsActionFlags()`와 `setIconifiedByDefault()`에 대해서도 주의깊게 살펴보기 바란다.

내비게이션 모드 종류 살펴보기

액션바는 안드로이드 3.0 이후부터는 매우 유용한 영역이다. 이 영역을 잘 활용하면 태블릿을 포함한 어떤 기기에서도 사용자가 해당 애플리케이션의 기능을 쉽게 다루도록 할 수 있다. 지금까지는 액션바에 액션 아이템(메뉴)만을 포함하고 있었지만, 이 섹션에서는 탭이나 리스트를 통해 사용자가 애플리케이션을 탐색할 수 있는 기능인 내비게이션 모드에 대해 살펴보겠다.

액션바에서 제공하는 내비게이션 모드에는 표준 모드, 탭 모드, 그리고 리스트 모드 이렇게 3가지가 있다.

(1) 표준 모드(NAVIGATION_MODE_STANDARD)

액션바에 탭이나 리스트를 포함하지 않은 내비게이션 모드를 말한다. 이 모드는 지금까지 살펴본 액션바의 형태를 가지며, 내비게이션 모드를 별도로 지정하지 않을 경우 자동으로 지정되는 모드다.

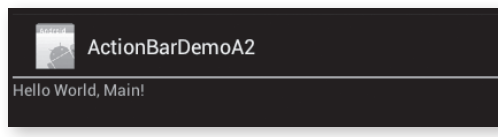


그림 5-21

(2) 탭 모드(NAVIGATION_MODE_TABS)

액션바에 탭 메뉴를 추가하여 사용자가 애플리케이션의 기능을 쉽게 활용할 수 있도록 해주는 내비

게이션 모드다. 이 모드를 사용하더라도 오른쪽 끝에 액션 아이템(메뉴)을 추가할 수 있는 것은 변함없다.

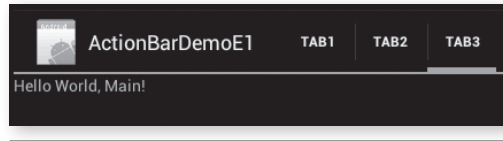


그림 5-22

(3) 리스트 모드(NAVIGATION_MODE_LIST)

액션바에 리스트 메뉴를 추가하여 사용자가 애플리케이션을 쉽게 탐색할 수 있도록 해주는 내비게이션 모드다. 이 모드에서도 당연히 오른쪽에 액션 아이템을 추가할 수 있다.

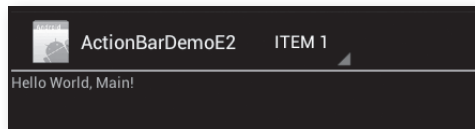


그림 5-23

내비게이션 모드를 설정하는 메소드는 ActionBar 클래스에 정의되어 있는 `setNavigationMode()` 메소드다. 이 메소드에 원하는 내비게이션 모드를 지정해서 호출하면 된다.

API

METHOD SUMMARY

android.app.ActionBar 클래스의 내비게이션 관련 메소드

void	<div> setNavigationMode(int mode) 액션바의 내비게이션 모드를 설정한다. </div> <div style="margin-top: 10px;"> 매개변수 mode NAVIGATION_MODE_STANDARD, NAVIGATION_MODE_LIST, NAVIGATION_MODE_TABS </div>
------	--

이 섹션에서는 표준 모드를 제외한 탭 모드와 리스트 모드에 대해서 살펴보도록 하겠다. 필자가 생각하기에 액션바의 핵심은 바로 이 내비게이션 모드이며, 이를 얼마나 잘 활용하냐에 따라 애플리케이션의 편의성이 달라진다고 생각한다.

탭(Tab) 모드 살펴보기

참고 프로젝트 ActionBarDemoE1

이 섹션에서는 액션바에 3개의 탭을 추가하고 탭을 클릭했을 때 Toast를 보이게 할 것이다. 액션 아이템을 액션바 오른쪽에 추가할 수도 있지만 예제가 복잡해질 수 있으므로 탭과 관련된 코드만 작성할 것이다. 일단 태블릿과 스마트폰 화면에서 탭이 어떻게 나오는지 보도록 하자.

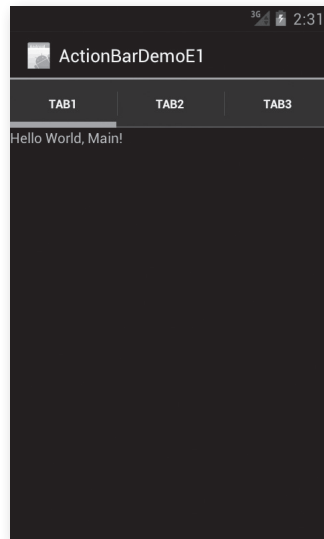


그림 5-24 액션바의 탭 모드 – WVGA(800*480)

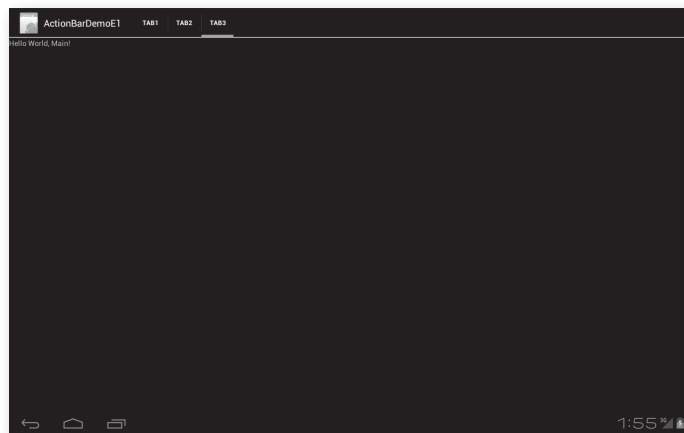


그림 5-25 액션바의 탭 모드 – WXGA(1280*800)

스마트폰 화면(WVGA)에서는 탭이 액션바 영역 아래에 표시되는 것을 알 수 있다. 이렇듯 태블릿과 폰 화면에서 표시되는 영역이 다르므로 애플리케이션 화면 기획 시에 이러한 부분들을 잘 확인해야 할 것이다.

이 예제는 액티비티의 자바 코드만 수정하면 되므로 기본 프로젝트를 생성해서 Main.java를 다음과 같이 수정하도록 하자.



CODE

Main.java

```
package com.androidside.actionbardemo1;

import android.app.ActionBar;
import android.app.ActionBar.Tab;
import android.app.ActionBar.TabListener;
import android.app.Activity;
import android.app.FragmentTransaction;
import android.os.Bundle;
import android.widget.Toast;
import com.androidside.actionbardemo1.R;

public class Main extends Activity implements TabListener {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        ActionBar actionBar = getActionBar();
        actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

        actionBar.addTab(actionBar.newTab().setText("Tab1").setTabListener(this));
        actionBar.addTab(actionBar.newTab().setText("Tab2").setTabListener(this));
        actionBar.addTab(actionBar.newTab().setText("Tab3").setTabListener(this));
    }

    @Override
    public void onTabReselected(Tab tab, FragmentTransaction ft) {
        Toast.makeText(this, tab.getText() + " onTabReselected", Toast.LENGTH_SHORT)
            .show();
    }

    @Override
    public void onTabSelected(Tab tab, FragmentTransaction ft) {
        Toast.makeText(this, tab.getText() + " onTabSelected", Toast.LENGTH_SHORT).show();
    }
}
```

```

    }

    @Override
    public void onTabUnselected(Tab tab, FragmentTransaction ft) {
        Toast.makeText(this, tab.getText() + " onTabUnselected", Toast.LENGTH_SHORT)
            .show();
    }
}

```

코드의 주요 부분을 하나씩 살펴보도록 하겠다.

actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);

현재의 액션바의 내비게이션 모드를 탭 모드(NAVIGATION_MODE_TABS)로 설정하겠다는 의미다. 여기서 지정한 탭 모드와 관련된 상수 NAVIGATION_MODE_TABS는 ActionBar 클래스에 선언되어 있다.

actionBar.addTab(actionBar.newTab().setText("Tab1").setTabListener(this));

addTab() 메소드는 새로 생성한 탭을 액션바의 탭으로 지정하겠다는 것이며, newTab() 메소드는 탭을 새로 생성하겠다는 것이다. 그리고 setTabListener() 메소드는 탭에 리스너를 등록하여 클릭 등의 이벤트를 처리하겠다는 것이다. 코드에 익숙하지 않을 경우 이 코드가 매우 복잡해 보일 수 있지만, 대부분의 개발자들은 코드를 이렇게 작성하므로 익숙해져야 할 것이다. 이 코드를 쉽게 풀어 작성하면 다음과 같다.

```

Tab tab1 = actionBar.newTab();
tab1.setText("Tab1");
tab1.setTabListener(this);
actionBar.addTab(tab1);

```



METHOD SUMMARY

android.app.ActionBar 클래스의 내비게이션 탭 생성 및 추가 메소드

ActionBar.Tab

newTab()

새로운 ActionBar.Tab을 생성하고 반환한다. 생성한 탭은 addTab() 메소드가 호출되기 전까지는 액션바에 추가되지 않는다.

반환

새로 생성한 ActionBar.Tab

void	addTab(ActionBar.Tab tab) 액션바에 지정된 tab을 추가한다. 매개변수 tab 액션바에 포함할 탭
------	---

public class Main extends Activity implements TabListener {

액션바에 탭을 추가하는 것이기 때문에 탭을 클릭했을 때 이를 감지해서 적당한 처리를 하는 것이 필요하다. 그렇기 때문에 TabListener 인터페이스의 메소드를 구현할 필요가 있다. TabListener를 implements로 선언하고, Main 클래스 내부에 onTabReselected(), onTabSelected(), 그리고 onTabUnselected() 메소드를 작성하면 된다. 이 메소드에 대한 설명은 다음 표를 참고하기 바란다.



METHOD SUMMARY

android.app.ActionBar 클래스의 내비게이션 탭 이벤트 메소드

void	onTabReselected(ActionBar.Tab tab, FragmentTransaction ft) 선택된 탭이 다시 선택되었을 때 호출된다. 매개변수 tab 다시 선택된 탭 ft 프래그먼트 처리 담당 클래스
void	onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) 탭이 선택된 상태가 되었을 때 호출된다. 매개변수 tab 선택된 탭 ft 프래그먼트 처리 담당 클래스
void	onTabUnselected(ActionBar.Tab tab, FragmentTransaction ft) 탭이 선택된 상태가 해지되었을 때 호출된다. 매개변수 tab 선택 해지된 탭 ft 프래그먼트 처리 담당 클래스

※ FragmentTransaction 클래스에 대해서는 프래그먼트 부분에서 살펴볼 것이므로 여기서는 이런 게 있구나 정도로 생각하기 바란다.

리스트(List) 모드 살펴보기

참고 프로젝트 ActionBarDemoE2

이 섹션에서는 액션바에 리스트를 추가하고 클릭했을 때 Toast 메시지를 화면에 출력하도록 하겠다. 다음 두 화면은 태블릿 크기와 스마트폰 크기에서 보여지는 액션바의 리스트 모양이다. 주의해서 봐야 할 부분은 리스트가 액션바에 표시되었다는 것이다.



그림 5-26 액션바의 리스트 모드 – WVGA(800*480)



그림 5-27 액션바의 리스트 모드 – WXGA(1280*800)

이렇게 만들기 위한 절차는 다음과 같다.

- (1) 리스트에 표현할 문자열 배열 작성 - /res/values/strings.xml
- (2) 문자열 배열로 SpinnerAdapter 생성
- (3) 액션바에 SpinnerAdapter 설정

가장 먼저 리스트에 표시할 문자열 배열을 생성해야 한다. 다음처럼 strings.xml에 <string-array>를 선언하면 된다.



CODE

문자열 XML - /res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="hello">Hello World, Main!</string>
    <string name="app_name">ActionBarDemoE2</string>

    <string-array name="action_list">
        <item>ITEM 1</item>
        <item>ITEM 2</item>
        <item>ITEM 3</item>
    </string-array>

</resources>
```

이제 선언된 문자열 배열을 액션바의 리스트로 표시할 액티비티를 작성하면 된다. 일단 전체 코드를 보도록 하자.



CODE

Main.java

```
package com.androidside.actionbardemoe2;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.SpinnerAdapter;
import android.widget.Toast;

public class Main extends Activity implements ActionBar.OnNavigationListener {
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    SpinnerAdapter spinnerAdapter = ArrayAdapter.createFromResource(this,
        R.array.action_list,
        android.R.layout.simple_spinner_dropdown_item);

    ActionBar actionBar = getActionBar();
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
    actionBar.setListNavigationCallbacks(spinnerAdapter, this);
}

@Override
public boolean onNavigationItemSelected(int itemPosition, long itemId) {
    String[] actions = getResources().getStringArray(R.array.action_list);

    Toast.makeText(this, actions[itemPosition], Toast.LENGTH_SHORT).show();

    return true;
}
}

```

주요 코드에 대해서 하나씩 살펴해보도록 하겠다.

```

SpinnerAdapter spinnerAdapter = ArrayAdapter.createFromResource(this,
    R.array.action_list,
    android.R.layout.simple_spinner_dropdown_item);

```

SpinnerAdapter를 /res/values/strings.xml에 선언한 문자열 배열인 action_list를 사용해서 생성한다. 이때 문자열 배열로 SpinnerAdapter를 생성해야 하므로 ArrayAdapter 클래스의 createFromResource() 메소드를 사용했으며, 안드로이드에서 기본으로 제공하는 레이아웃을 사용하기 위해 android.R.layout.simple_spinner_dropdown_item을 지정했다.

```

actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);

```

내비게이션 모드를 리스트 형태로 설정한다.

```
actionBar.setListNavigationCallbacks(spinnerAdapter, this);
```

spinnerAdapter를 액션바에 설정하고 관련된 이벤트를 현재 클래스에서 처리할 수 있게 한다. 이렇게 this를 선언하기 위해서는 현재 클래스(액티비티)가 ActionBar.OnNavigationListener를 구현하고 있어야 한다.

```
public class Main extends Activity implements ActionBar.OnNavigationListener {
```

SpinnerAdapter의 이벤트를 액션바에서 처리할 수 있도록 ActionBar.OnNavigationListener를 구현하고 관련 메소드인 onNavigationItemSelected()를 작성한다.



METHOD SUMMARY

android.app.ActionBar.OnNavigationListener 인터페이스의 메소드

boolean	onNavigationItemSelected(int itemPosition, long itemId) 내비게이션 아이템을 선택할 때 호출된다.
매개변수	
itemPosition	선택한 아이템의 위치
itemId	선택한 아이템의 아이디(ID)
반환	
	이벤트의 처리 여부(처리하면 true, 그렇지 않으면 false)



TIP & TECH

액션바의 리스트 모드에서 리스트의 문자열이 매우 길 때

일반 스마트폰 화면에서 리스트는 액션바의 아이콘과 같은 라인에 표시되는 반면에 탭은 액션바의 아래에 표시된다. 그런데 만약 리스트의 내용이 매우 길 경우에는 어떻게 될까? 매우 길기 때문에 탭처럼 액션바의 아래에 표시가 될까? 실제로 문자열을 길게 작성해서 실행해보면 액션바의 제목이 사라지고 리스트가 보이는 것을 알 수 있다. 또한 리스트의 내용이 매우 길 때는 일부분만 보인다는 것도 알 수 있다. 이러한 표시 방식을 이해하고 있어야 향후 애플리케이션 화면 기획을 할 때 시행착오를 조금이나마 줄일 수 있을 것이다.

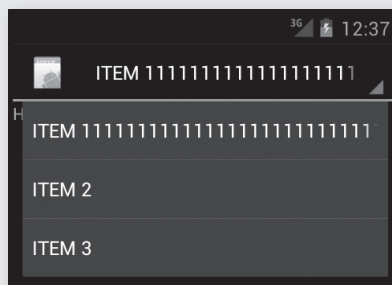


그림 5-28