**1.     구글 맵 Key 받기**

**Google Maps 데이터에 접근하는 Android에 어플리케이션을 제작하기 위해서 Google Maps service에 등록 및 약관 동의 절차를 거쳐야 한다.**

**Windows XP 기준으로 설명하겠다.**

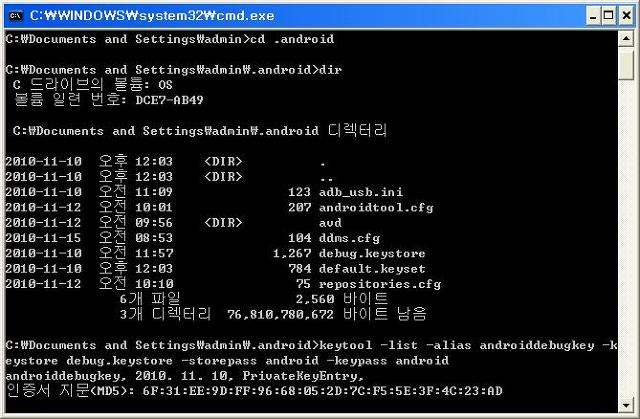
**1)     SDK 디버그 서명 증명서의 MD5 핑거프린트 확인하기**

**디버그 서명 증명서는 Android SDK가 자동 생성하며 개발 환경의 OS 마다 다른 경로로 생성 된다.**

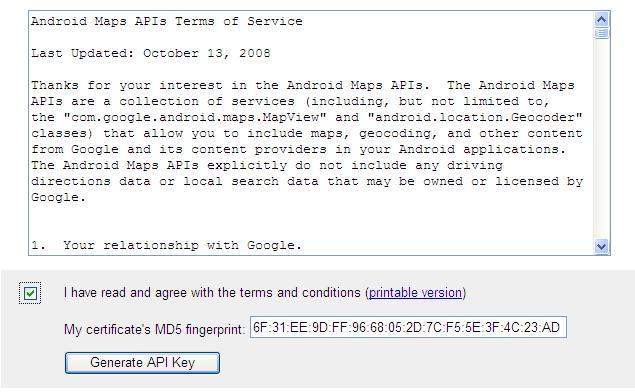
|  |
| --- |
| Win Vista: C:\Users\%username%\.android\debug.keystore  Win XP: C:\Documents and Settings\%username%\.android\debug.keystore  OS X and Linux: ~/.android/debug.keystore |

**2)     명령프롬프트 창에서 인증서 지문을 발급 받는다.**

**keytool -list -alias androiddebugkey -keystore debug.keystore -storepass android -keypass android**

****

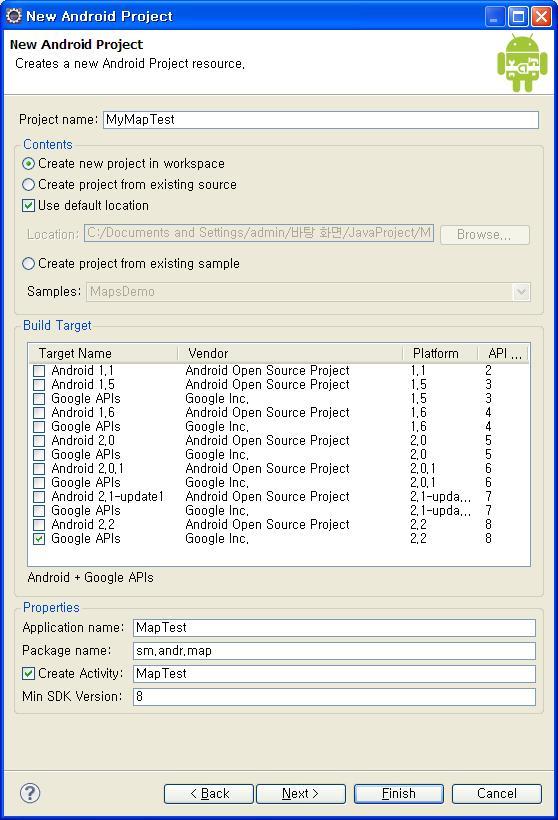
**3)     인증서 지문을** [**http://code.google.com/intl/ko/android/maps-api-signup.html**](http://code.google.com/intl/ko/android/maps-api-signup.html) **사이트에 접속하여 등록한다.**

****

**4)     마지막 단계로 Key를 확인한다.**

****

**2.     안드로이드 프로젝트 생성하기**

****

**1)     MyMapTest의 AndroidManifest.xml 편집 하기**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*  package=*"sm.andr.map"*  android:versionCode=*"1"*  android:versionName=*"1.0"*>  <application android:icon=*"@drawable/icon"* android:label=*"@string/app\_name"*>  <activity android:name=*".MapTest"*  android:label=*"@string/app\_name"*>  <intent-filter>  <action android:name=*"android.intent.action.MAIN"* />  <category android:name=*"android.intent.category.LAUNCHER"* />  </intent-filter>  </activity>  <uses-library android:name=*"com.google.android.maps"* />  </application>  <uses-sdk android:minSdkVersion=*"8"* />  <uses-permission android:name=*"android.permission.INTERNET"* />    </manifest> |
| 밑줄 쳐진 두 코드는 Standard Android library에는 Maps library가 없으므로 별도의 설정이 필요하며, 인터넷에 접속하여 Maps data를 조회하기 위해 InterNet Permission을 지정 한다. |

**2)     레이아웃 작성하기**

**MyMapTest의 res -> layout -> main.xml에 다음과 같은 코드를 추가 한다.**

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:orientation=*"vertical"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"*  >    <com.google.android.maps.MapView  android:id=*"@+id/mapview"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"*  android:enabled=*"true"*  android:clickable=*"true"*  android:apiKey=*"*0mw9Gb6hyd3M2zsisai4MltNSLEa8ON7XAdWedA*"*  />  </LinearLayout> |
| 코드를 작성하고 apiKey에는 처음에 발급 받은 키를 적어 준다. |

**3)     Java 코드 작성하기**

**MapActivity, MapView를 import! 하고 MapActivity를 상속 한다.**

**MapActivity 클래스는 MapView에 지도 표시를 관리하는 역할을 한다.**

|  |
| --- |
| **package** bit.andr.map;  **import!** com.google.android.maps.GeoPoint;  **import!** com.google.android.maps.MapActivity;  **import!** com.google.android.maps.MapView;  **import!** android.os.Bundle;    **public** **class** MyMap **extends** MapActivity {  **private** MapView mv;  /\*\* Called when the activity is first created. \*/  @Override  **public** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*main*);    initMapView();  }    **private** **void** initMapView() {  // **TODO** Auto-generated method stub  mv = (MapView)findViewById(R.id.*mapview*);  mv.setBuiltInZoomControls(**true**);  mv.setSatellite(**false**);  }  @Override  **protected** **boolean** isRouteDisplayed() {  // **TODO** Auto-generated method stub  **return** **true**;  }  } |
| 위의 코드는 구글 맵전체를 안드로이드에 출력해 주는 역할을 한다. isRouteDisplayed() 메소드는 MapActivity의 추상 메소드이므로 반드시 구현해야 한다. 이 메소드는 애플리케이션에 라우트 정보를 표시할 경우 true를 반환하고, 그렇지 않은 경우에는 false를 반환한다. |

**4)     실행화면**

|  |  |
| --- | --- |
| http://cfile299.uf.daum.net/image/122EC3314CE14DF01F548B | |
| MyMap01.java | |
| **public** **class** MyMap01 **extends** MapActivity {    **private** MapView mapView=**null**;    MapController mapController = **null**;  MyLocationOverlay whereAmI = **null**;    @Override  **public** **void** onCreate(Bundle savedInstanceState) {  **super**.onCreate(savedInstanceState);  setContentView(R.layout.*main*);    mapView = (MapView)findViewById(R.id.*geoMap*);  mapView.setBuiltInZoomControls(**true**);    Drawable marker=getResources().getDrawable(R.drawable.*pos*);  marker.setBounds(0,0,marker.getIntrinsicWidth()  ,marker.getIntrinsicHeight());    mapController = mapView.getController();  mapController.setZoom(10);    whereAmI = **new** MyLocationOverlay(**this**, mapView);  mapView.getOverlays().add(whereAmI);    MyOverlayTest funPlaces = **new** MyOverlayTest(marker);  mapView.getOverlays().add(funPlaces);  mapView.postInvalidate();    }  @Override  **protected** **boolean** isLocationDisplayed() {    **return** whereAmI.isMyLocationEnabled();  }  @Override  **protected** **boolean** isRouteDisplayed() {  // **TODO** Auto-generated method stub  **return** **false**;  }    @Override  **public** **void** onResume()  {  **super**.onResume();  whereAmI.enableMyLocation();  whereAmI.runOnFirstFix(**new** Runnable() {  **public** **void** run() {  mapController.setCenter(whereAmI.getMyLocation());  }  });  }  @Override  **public** **void** onPause()  {  **super**.onPause();  whereAmI.disableMyLocation();  }    } | |

마커를 집어 넣고 라인을 그린다.

|  |
| --- |
| MyOverlayTest.java |
| **public** **class** MyOverlayTest **extends** ItemizedOverlay {    **private** List<OverlayItem> locations = **new** ArrayList<OverlayItem>();  **private** Drawable marker;  GeoPoint disney[];    **public** MyOverlayTest(Drawable \_marker) {    **super**(\_marker);  **this**.marker = \_marker;  **float** num=37.522006f;  **float** num1=-233.004080f;  disney = **new** GeoPoint[5];    **for**(**int** i=0;i<disney.length;i++)  {  //관심 지점들 생성  disney[i]=**new** GeoPoint((**int**)(num\*1000000),  (**int**)(num1\*1000000));  locations.add(**new** OverlayItem(disney[i],"서울","서울"));  num= num+0.05f;  num1= num1-0.1f;  }  populate();  }    @Override  **public** **void** draw(Canvas canvas,MapView mapView, **boolean** shadow)  {  **super**.draw(canvas, mapView, shadow);  *boundCenterBottom*(marker);    Paint paint1 = **new** Paint();  paint1.setARGB(255, 0, 0, 0);  Point pixPoint = **new** Point();  Point pixPoint2 = **new** Point();    **for**(**int** i=0;i<disney.length-1;i++)  {  // 지리좌표를 화면상의 픽셀좌표로 변환  mapView.getProjection().toPixels(disney[i], pixPoint);  mapView.getProjection().toPixels(disney[i+1], pixPoint2);  //라인 그리기  canvas.drawLine(pixPoint.x,pixPoint.y,pixPoint2.x,pixPoint2.y, paint1);  }  mapView.getProjection().toPixels(disney[0], pixPoint);  mapView.getProjection().toPixels(disney[disney.length-1], pixPoint2);    //Text집어 넣기  canvas.drawText("시작점", pixPoint.x-12, pixPoint.y + 23, paint1);  canvas.drawText("끝점", pixPoint2.x-12, pixPoint2.y + 23, paint1);  }    @Override  **protected** OverlayItem createItem(**int** i) {  // **TODO** Auto-generated method stub  **return** locations.get(i);  }    @Override  **public** **int** size() {  **return** locations.size();  }    } |

발급받은 키를 사용하여 구글맵API를 사용한다.

|  |
| --- |
| Main.xml |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <LinearLayout xmlns:android=*"http://schemas.android.com/apk/res/android"*  android:orientation=*"vertical"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"*  >  <TextView  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"wrap\_content"*  android:text=*"@string/hello"*  />  <com.google.android.maps.MapView  android:id=*"@+id/geoMap"*  android:clickable=*"true"*  android:layout\_width=*"fill\_parent"*  android:layout\_height=*"fill\_parent"*  android:apiKey=*"0ftmNTVrwlNEsrlhjKUohrWl0pGbfb3VYuVrwsg"* />  </LinearLayout> |

|  |
| --- |
| AndroidManifest.xml |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*  package=*"kim.MyMap01"*  android:versionCode=*"1"*  android:versionName=*"1.0"*>  <application android:icon=*"@drawable/icon"* android:label=*"@string/app\_name"*>  <uses-library android:name=*"com.google.android.maps"* />  <activity android:name=*".MyMap01"*  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>  <uses-permission android:name=*"android.permission.ACCESS\_FINE\_LOCATION"* />  <uses-permission android:name=*"android.permission.ACCESS\_COARSE\_LOCATION"* />  <uses-permission android:name=*"android.permission.INTERNET"*/>  <uses-sdk android:minSdkVersion=*"8"* />  </manifest> |

|  |
| --- |
| 실행 이미지 |
| http://cfile272.uf.daum.net/image/12564B364D08EAD72F1DF1 |

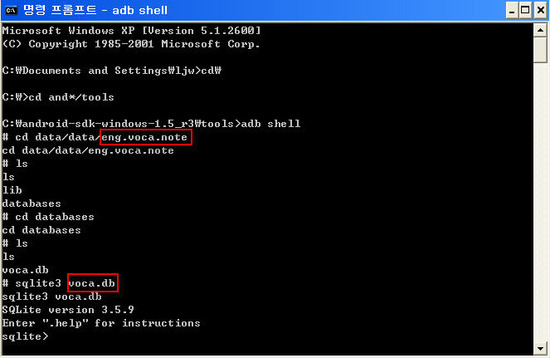
mapview위에 mark를 표시하는 방법은 Overlay를 이용한 것이다.

그런데, 안드로이드 상에서 MyLocationOverlay라는 특수한 overlay를 제공해준다. 이름에서 알 수 있듯이 현재 target의 위치를 mapview위에 띄워주는 overlay이다. 자신의 위치를 나타내고, 위치가 갱신이 되면 이동한 경로를 mapview에 그리고, 이동 거리를 계산하는 코드는 다음과 같다.

|  |
| --- |
| class MyOverlay extends MyLocationOverlay {  Location mMyBeforeLocation;  Location mMyCurrentLocation;  Path mPath;  Paint mPaint;  ArrayList mMyPathLocationArray;  Context mContext;  MapView mMapView;  float mMyDistance = 0;  public MyOverlay(Context context, MapView mapView)   {   super(context, mapView);      mPath = new Path();   mPath.reset();   mPaint = new Paint();      mPaint.setAntiAlias(true);      mPaint.setDither(true);      mPaint.setColor(Color.RED);      mPaint.setStyle(Paint.Style.STROKE);      mPaint.setStrokeJoin(Paint.Join.ROUND);      mPaint.setStrokeCap(Paint.Cap.ROUND);      mPaint.setStrokeWidth(3);      mPaint.setTextSize(20);      mPaint.setStyle(Paint.Style.FILL);      mMyPathLocationArray = new ArrayList();            mContext = context;      mMapView = mapView;  }    public void onLocationChanged(Location location)  {   super.onLocationChanged(location);   Double latitude = location.getLatitude();   Double longitude = location.getLongitude();   Geocoder gc = new Geocoder(mContext, Locale.getDefault());   StringBuilder sb = new StringBuilder();   try {    List  *addresses = gc.getFromLocation(latitude, longitude, 10);*         if(addresses.size() > 0)    {     Address address = addresses.get(0);     for(int i = 0 ; i <= address.getMaxAddressLineIndex() ; i++)     {      if(i > 0)       sb.append(" ");      sb.append(address.getAddressLine(i));     }    }    else    {     sb.append(mContext.getString(R.string.unknown\_address));    }    ((MapTest)mContext).mMyLocationEditText.setText(sb.toString()); //현재 위치를 주소로 변환하여 text로 찍어준다.   } catch (IOException e) {    e.printStackTrace();   }      mMyBeforeLocation = mMyCurrentLocation;   mMyCurrentLocation = location;   if(mMyBeforeLocation != null && mMyCurrentLocation != null && mMyBeforeLocation != mMyCurrentLocation)   {    mMyPathLocationArray.add(new MyPathLocation(mMyBeforeLocation, mMyCurrentLocation)); //위치가 바뀔때마다 path를 추가해준다.        mMyDistance += mMyCurrentLocation.distanceTo(mMyBeforeLocation);  //이동한 거리를 계산한다.        ((MapTest)mContext).mMyLocationEditText.setText(String.valueOf(mMyDistance));   }   else     return;     }    public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when)  { //draw함수는 무한 반복 호출되는 callback이다   if(mMyBeforeLocation != null && mMyCurrentLocation != null)   {    mPath.reset();  //비어있는 path를 만들어 mapview위를 깨끗하게 초기화 한다.    canvas.drawPath(mPath, mPaint);    updatePath();  //위치가 바뀔때마다 추가해두었던 path 배열을 그리기위한 준비.    mPaint.setStyle(Paint.Style.STROKE);    canvas.drawPath(mPath, mPaint);   }   return super.draw(canvas, mapView, shadow, when);  }    public void updatePath()  {   for(int i = 0 ; i < mMyPathLocationArray.size() ; i++)   {    Point startPoint = new Point();    Point endPoint = new Point();    MyPathLocation temp = mMyPathLocationArray.get(i);    mMapView.getProjection().toPixels(new GeoPoint((int)(temp.mMyBeforeLocation.getLatitude()\*1E6), (int)(temp.mMyBeforeLocation.getLongitude()\*1E6)), startPoint);    mMapView.getProjection().toPixels(new GeoPoint((int)(temp.mMyCurrentLocation.getLatitude()\*1E6), (int)(temp.mMyCurrentLocation.getLongitude()\*1E6)), endPoint);        Path p = new Path();    p.reset();    p.moveTo(startPoint.x, startPoint.y);    p.lineTo(endPoint.x, endPoint.y);    mPath.addPath(p);   }  } }  class MyPathLocation {  public Location mMyBeforeLocation;  public Location mMyCurrentLocation;    public MyPathLocation(Location myBeforeLocation, Location myCurrentLocation)  {   mMyBeforeLocation = myBeforeLocation;   mMyCurrentLocation = myCurrentLocation;  } } |

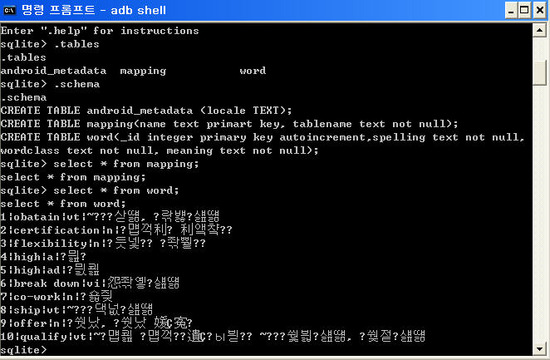
코드상으로 건드린 안드로이드 DB... 실제로 테이블들을 확인할 수 있는 방법은?? cmd 창에서 adb를 이용해 확인할 수 있다. adb는 android debug bridge 로써 안드로이드 디버깅 인터페이스다. 이후 명령을 수행하기 위해선 반드시 안드로이드 에뮬이 켜져 있어야 한다.

[시작] -> [실행] -> cmd 입력 후 확인 으로 cmd 창을 켜자.



빨갛게 표시된 부분은 각각 살펴볼 프로젝트의 패키지 명과 만들어놓은 DB 파일 이름이다.

목록을 보고싶으면 dir과 ls 명령을 사용하면 되겠다.



대충 요런식으로 사용하면 되겠다. 이외에도 DB관련 쿼리문을 수행할 수 있다.

**+) 유저 디비의 위치 : data/data/"어플 패키지명"/databases/"디비이름.db"**

이예제는 MapView  + LocationManager + Overlay  + GeoCoder 를 활용한 예제입니다.

실행결과:



package kims.android.example;

import java.io.IOException;

import java.util.List;

import java.util.Locale;

import android.content.Context;

import android.graphics.Bitmap;

import android.graphics.BitmapFactory;

import android.graphics.Canvas;

import android.graphics.Color;

import android.graphics.Paint;

import android.graphics.Point;

import android.location.Address;

import android.location.Criteria;

import android.location.Geocoder;

import android.location.Location;

import android.location.LocationListener;

import android.location.LocationManager;

import android.os.Bundle;

import android.util.Log;

import com.google.android.maps.GeoPoint;

import com.google.android.maps.MapActivity;

import com.google.android.maps.MapController;

import com.google.android.maps.MapView;

import com.google.android.maps.Overlay;

import com.google.android.maps.Projection;

public class MapTestActivity extends MapActivity

      implements LocationListener {

MapView mapView;

MapController controller;

LocationManager manager;

String provider;

Geocoder coder;

String TAG="MapTestActivity";

@Override

protected void onCreate(Bundle arg0) {

super.onCreate(arg0);

setContentView(R.layout.map\_layout);

mapView=(MapView)findViewById(R.id.mapView);

controller=mapView.getController();

mapView.setBuiltInZoomControls(true);

checkProvider();

coder = new Geocoder(getApplicationContext(), Locale.KOREA);

}

@Override

protected void onResume() {

   super.onResume();

   // 마지막으로 얻은 위치정보가 있으면 보기

    Location loc = manager.getLastKnownLocation(LocationManager.GPS\_PROVIDER);

    if (loc != null) {

       onLocationChanged(loc);

    }

    // GPS Provider로 부터 위치 정보를 수신하게 되면 호출될 LocationListener 등록

    manager.requestLocationUpdates(LocationManager.GPS\_PROVIDER,0,0,this);

}

@Override

protected void onPause() {

       super.onPause();

       // 위치정보 통지 등록 취소

       manager.removeUpdates(this);

}

@Override

protected boolean isRouteDisplayed() {

        return false;

}

// 지도상에 화살표 이미지와 텍스트 문자열을 그리기위한 메서드

public void setOverlay(GeoPoint p, String addr) {

      Bitmap icon = BitmapFactory.decodeResource(getResources(),R.drawable.pin\_spot2);

      IconOverlay overlay = new IconOverlay(icon, p, addr);

      List<Overlay> overlays = mapView.getOverlays();

      // overlays.clear(); //기존 overlay되어 있는 정보를 제거한다.

      overlays.add(overlay);

}

// GeoCoder를 이용해서 수신된 위치정보에 대한 주소 정보를 찾아서 문자열로

// 반환하는 메서드

private  String showLocationName(Location loc) {

      double latitude = loc.getLatitude();

      double longitude = loc.getLongitude();

      StringBuffer buff = new StringBuffer();

      try {

          List<Address> addrs =

          coder.getFromLocation(latitude, longitude, 1);

          for (Address addr : addrs) {

              int index = addr.getMaxAddressLineIndex();

              for (int i = 0; i <= index; ++i) {

                     buff.append(addr.getAddressLine(i));

                     buff.append(" ");

               }

              buff.append("\n");

           }

        } catch (IOException e) {

               Log.e(TAG, e.toString());

      }

      return buff.toString();

}

// 가장 베스트하게 사용가능한 Provider를 검색한다.

public void checkProvider() {

       manager=(LocationManager)getSystemService(Context.LOCATION\_SERVICE);

       Criteria cta=new Criteria();

       cta.setAccuracy(Criteria.ACCURACY\_FINE);

       cta.setPowerRequirement(Criteria.POWER\_HIGH);

       cta.setAltitudeRequired(false);

       cta.setSpeedRequired(false);

       cta.setCostAllowed(false);

       provider=manager.getBestProvider(cta, true);

}

// 위치정보가 변경되면 호출된다.

public void onLocationChanged(Location location) {

          GeoPoint gp = new GeoPoint((int)(location.getLatitude()\*1E6),

                                                     (int)(location.getLongitude()\*1E6));

          controller.animateTo(gp);

          setOverlay(gp,showLocationName(location));

}

// 위치정보를 제공하는 프로바이더가 disable되었을때 호출된다.

public void onProviderDisabled(String provider) {

}

// 위치정보를 제공하는 프로바이더가 enable되었을때 호출된다.

public void onProviderEnabled(String provider) {

}

// 위치정보를 제공하는 프로바이더가 enable 또는 disable되었을때 호출된다.

public void onStatusChanged(String provider, int status,

Bundle extras) {

}

// 지도위에 이미지나 텍스트를 출력하기 위한 클래스

private class IconOverlay extends Overlay {

      Bitmap mIcon;

      GeoPoint mPoint;

      String addr;

      int mOffsetX;

      int mOffsetY;

      IconOverlay(Bitmap icon, GeoPoint initial, String addr) {

          mIcon = icon;

          mOffsetX = 0 - icon.getWidth() / 2;

          mOffsetY = 0-icon.getHeight();

          mPoint = initial;

          this.addr=addr;

       }

// 지도를 탭하면 호출된다.

@Override

public boolean onTap(GeoPoint point, MapView mapView) {

        mPoint = point;

        return super.onTap(point, mapView);

}

// 지도를 그릴 때, shadow = true, shadow = false를 2 번 호출

@Override

public void draw(Canvas canvas, MapView mapView, boolean shadow) {

     super.draw(canvas, mapView, shadow);

     if (!shadow) {

      // 지도상 위치와 그림을 그릴  Canvas상의  좌표 변환을 위한 객체

     Projection projection = mapView.getProjection();

     Point point = new Point();

     projection.toPixels(mPoint, point);

     Paint paint=new Paint();

     paint.setAntiAlias(true);

     paint.setTextSize(14);

     paint.setColor(Color.RED);

     canvas.drawText(addr, point.x, point.y-20, paint);   // 지정된 좌표에 문자를 출력

     point.offset(mOffsetX, mOffsetY);

     canvas.drawBitmap(mIcon, point.x, point.y, null);  //지정된 좌표에 화살표 이미지 출력

     }

  }

}//IconOverlay끝

}

//////////////////////////////////////////////////////////////////////////////////////////////////////

**SMS 문자가 수신되면 현재 자신의 위치정보(위도,경도)를 서버로 전송하는 예제**

여기에서는 SMS 문자가 수신되면 BroadcastRecevicer가 수신하여 현재 위치 정보를

점검하여 로그를 출력하는 작업까지만 설명하고 있다.

커맨트에 있는것 처럼 서버로 전송하는 작업이 필요하다면 체크된 위도,경도 정보를

HttpClient 나 URL 을 사용하여 서버에 전송하는 코드를 작성하면 된다.

이런 경우는 화면을 가지고 있지 않으므로 굳이 비동기식 서버 처리가 요구되지 않음으로

URL 을 이용하여 데이터를 전송하는것이 더 적절할 것 같다.

package kim.android.test;

import android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.location.Criteria;

import android.location.Location;

import android.location.LocationListener;

import android.location.LocationManager;

import android.os.Bundle;

import android.telephony.SmsMessage;

import android.util.Log;

import android.widget.Toast;

public class SmsReceiver extends BroadcastReceiver {

            Location location;

            LocationManager locationManager;

            // SMS가 수신되면 호출되는 메서드이다.

            @Override

            public void onReceive(Context context, Intent intent) {

               Bundle bundle = intent.getExtras();

               SmsMessage[] msgs = null;

               String number="";

               String message="";

               if (bundle != null) {

                   Object[] pdus = (Object[]) bundle.get("pdus");

                   msgs = new SmsMessage[pdus.length];

                   for (int i=0; i<msgs.length; i++){

                         msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);

                         // SMS를 보낸 전화번호를 얻어온다.

                         number=msgs[i].getOriginatingAddress();

                         // SMS 내용

                         message= msgs[i].getMessageBody().toString();

                         //Log.i("SmsReceiver","number: "+number+ "  message: "+message);

                       // 발신자와 발신메시지를 체크하여 위치정보를 요청하는

                       // 문자메시지이면 현재 위치정보를 전송한다.

                       if ( number.equals("01011112222") &&

                                      message.equals("position check")) {

                          Log.i("SmsReceiver","call sendMyLocation(context)");

                          checkMyLocation(context);

**// 이부분에 서버에 위도,경도값을 전송하는 코드를 작성하면 됩니다.**

                         Log.i("SmsReceiver","Current Location>>  latitude: "+

                                      location.getLatitude()+"  longitudeL "+

                                      location.getLongitude());

                       }

               }

          }

    }

**// 현재 위치 정보를 체크하는 메서드**

**// Criteria 클래스를 사용하여 현재 사용가능한 Provider중 가장**

**// 최적의 Provider를 선택하여 사용하도록 설정**

            public void checkMyLocation(Context context) {

                            locationManager =

                                                (LocationManager)context.getSystemService(Context.LOCATION\_SERVICE);

                                    Criteria criteria = new Criteria();

                                    criteria.setAccuracy(Criteria.ACCURACY\_FINE);       // 세밀한 정확도

                                    criteria.setPowerRequirement(Criteria.POWER\_HIGH);  // 전원 소비량

                                    criteria.setAltitudeRequired(false);             // 고도, 높이 값을 얻어 올지를 결정

                                    criteria.setSpeedRequired(false);             //속도

                                    criteria.setCostAllowed(false);                   //위치 정보를 얻어 오는데 들어가는 금전적 비용

                                    String provider = locationManager.getBestProvider(criteria, true);

                                    // 에뮬은 DDMS 작동보다  listener를 먼저 등록해야

                                    // 최초 DDMS의 변화를 받아들일수 있다.

                                    // 10초간격, 100미터 간격 이 생기면 이벤트가 발생했다고 감지

                                    locationManager.requestLocationUpdates(

                                                            provider, 10000, 100, new MyLocationListener());

                                    if (provider == null) {

                                                            //GPS제공자가 없으면 기지국에서라도 Location 정보 얻어옴.

                                                            Toast.makeText(context, "GPS 제공자가 없음!!", 2000).show();

                                                            provider = LocationManager.NETWORK\_PROVIDER;

                                                            location = locationManager.getLastKnownLocation( provider );

                                    }

                                    location = locationManager.getLastKnownLocation( provider );

                                    if (location == null){

                                                try {

                                                            Toast.makeText(context,

                                                                                   "에뮬레이터 GPS를 작동해주세요", 10000).show();

                                                            Thread.sleep(10000);  //에뮬레이터 DDMS 작동시간을 위한 인터벌

                                                } catch (InterruptedException e) {

                                                            // TODO Auto-generated catch block

                                                            e.printStackTrace();

                                                }

                                        location = locationManager.getLastKnownLocation( provider );

           }

   }

            // Location정보가 변경되는 이벤트가 발생을 처리한다.

            private class MyLocationListener implements LocationListener {

                        public void onLocationChanged(Location location) {

                                    // TODO Auto-generated method stub

                                    SmsReceiver.this.location = location;

                        }

                        public void onProviderDisabled(String provider) {

                                    // TODO Auto-generated method stub

                        }

                        public void onProviderEnabled(String provider) {

                                    // TODO Auto-generated method stub

                        }

                        public void onStatusChanged(String provider, int status, Bundle extras) {

                                    // TODO Auto-generated method stub

                        }

            }

}

**AndroidManifest.xml 파일 설정파일에 추가해 주어야 할 내용들**

 <**receiver** android:name=".SmsReceiver">

            <intent-filter>

                <action android:name="android.provider.Telephony.SMS\_RECEIVED"/>

            </intent-filter>

 <**/receiver**>

<**uses-permission** android:name="android.permission.**ACCESS\_FINE\_LOCATION**" />

<**uses-permission** android:name="android.permission.**RECEIVE\_SMS**" />

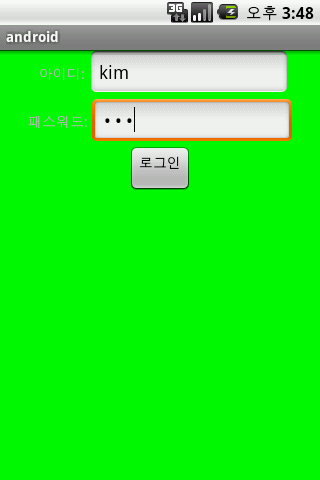
<**uses-permission** android:name="android.permission**.INTERNET"**/>

///////////////////////////////////////////////////////////////////////////////////////////////////////

HttpClient의 HttpPost 메서드를 사용하여 로그인을 처리한 예제.

**[실행결과]**

로그인이 성공한 경우 화면                     로그인이 실패한 경우 화면

**[login\_layout.xml]**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <LinearLayout      xmlns:android="<http://schemas.android.com/apk/res/android>"      android:id="@+id/layout01"      android:layout\_width="fill\_parent"      android:layout\_height="fill\_parent"      android:orientation="vertical"      android:visibility="visible">     <LinearLayout          android:orientation="horizontal"          android:layout\_width="fill\_parent"          android:layout\_height="wrap\_content"          android:gravity="center\_horizontal">         <TextView android:text="  아이디: "               android:id="@+id/TextView02"               android:layout\_width="wrap\_content"               android:layout\_height="wrap\_content"/>         <EditText android:text=""               android:id="@+id/id"               android:layout\_width="200px"               android:layout\_height="wrap\_content"/>      </LinearLayout>       <LinearLayout              android:orientation="horizontal"              android:layout\_width="fill\_parent"              android:layout\_height="wrap\_content"              android:gravity="center\_horizontal">         <TextView android:text="패스워드: "               android:id="@+id/TextView01"               android:layout\_width="wrap\_content"               android:layout\_height="wrap\_content"/>        <EditText android:text=""               android:id="@+id/passwd"               android:layout\_width="200px"               android:layout\_height="wrap\_content"              **android:password="true**"/>       </LinearLayout>   <LinearLayout       android:orientation="horizontal"     android:layout\_width="fill\_parent"     android:layout\_height="wrap\_content"     android:gravity="center\_horizontal">  <Button android:text="로그인"    android:id="@+id/loginButton"    android:layout\_width="wrap\_content"    android:layout\_height="wrap\_content"   android:gravity="center\_horizontal"/>     </LinearLayout>  </LinearLayout> |

**[LoginActivity.java]**

|  |
| --- |
| package kim.android.test;    import java.io.IOException;   import java.io.InputStream;   import java.io.InputStreamReader;   import java.util.ArrayList;    import org.apache.http.HttpEntity;   import org.apache.http.HttpResponse;   import org.apache.http.NameValuePair;   import org.apache.http.client.ClientProtocolException;   import org.apache.http.client.HttpClient;   import org.apache.http.client.ResponseHandler;   import org.apache.http.client.entity.UrlEncodedFormEntity;   import org.apache.http.client.methods.HttpPost;   import org.apache.http.impl.client.DefaultHttpClient;   import org.apache.http.message.BasicNameValuePair;   import org.apache.http.params.HttpConnectionParams;   import org.apache.http.params.HttpParams;   import org.xmlpull.v1.XmlPullParser;   import org.xmlpull.v1.XmlPullParserFactory;    import android.app.Activity;   import android.app.ProgressDialog;   import android.graphics.Color;   import android.os.Bundle;   import android.os.Handler;   import android.os.Message;   import android.util.Log;   import android.view.View;   import android.view.View.OnClickListener;   import android.widget.Button;   import android.widget.EditText;   import android.widget.LinearLayout;   import android.widget.Toast;      public class LoginActivity extends Activity implements OnClickListener {     private EditText id;     private EditText passwd;     private ProgressDialog pDialog;     private LinearLayout layout01;     private Button button;        @Override     protected void **onCreate**(Bundle savedInstanceState) {      // TODO Auto-generated method stub          super.onCreate(savedInstanceState);          setContentView(R.layout.login\_layout);          id=(EditText)findViewById(R.id.id);          passwd=(EditText)findViewById(R.id.passwd);          button=(Button)findViewById(R.id.loginButton);          button.setOnClickListener(this);          layout01=(LinearLayout)findViewById(R.id.layout01);      }       @Override    public void **onClick**(View v) {       // TODO Auto-generated method stub       loginProcess();      //로그인 버튼이 클릭되면 로그인 처리를 시작한다.     }  **// 네트웍 처리결과를 화면에 반영하기 위한 안드로이드 핸들러**     // responseHandler에 의해 처리된 결과가 success인 경우 바탕화면을 초록색으로 바꾸고     // 로그인이 성공했다는 메시지를 토스트로 출력     // 로그인이 실패한 경우 바탕화면을 빨강색으로 바꾸고 로그인실패 메시지를 토스트로 출력    private final Handler **handler** = new Handler() {       @Override       public void **handleMessage**(Message msg) {             pDialog.dismiss();             String result=msg.getData().getString("RESULT");             if ( result.equals("success") ) {                   layout01.setBackgroundColor(Color.GREEN);                   Toast.makeText(LoginActivity.this, "성공적으로 로그인하였습니다.",                                          Toast.LENGTH\_LONG).show() ;                  } else {                   layout01.setBackgroundColor(Color.RED);                   Toast.makeText(LoginActivity.this, "로그인 실패",                                          Toast.LENGTH\_LONG),show() ;                  }        }      };  **//서버에서 전송된 XML 데이터를 파싱하기 위한 메서드**      //이 예제에서는 서버에서 로그인이 성공하는 경우(id=kim&passwd=111)하는 경우 <result>success</result>      //실패하는 경우 <result>failed</result>를 반환하도록 설정해 두었다.      public String **parsingData**(InputStream input){         String result=null;         try {              XmlPullParserFactory factory= XmlPullParserFactory.newInstance();              XmlPullParser parser = factory.newPullParser();              parser.setInput(new InputStreamReader(input));              while ( parser.next() != XmlPullParser.END\_DOCUMENT) {                  String name=parser.getName();                   if ( name != null && name.equals("result"))                          result=parser.nextText();               }          }catch(Exception e){e.printStackTrace();}          return result;      }  **//로그인 버튼이 클릭되면 수행되는 메서드**  //  responseHandler는 Http요청에 대한 HttpResponse가 반환되면 결과를 처리하기 위한       //  콜백메서드를 정의하고 있는 객체이다.       //  Response를 받게 되면 parsingData()메서드를 호출하여 서버로 부터 받은 XML 파일을 처리하여       // 그결과를 result 문자열로 반환받는다.       // 이렇게 반환받은 result문자열을 화면에 반영하기위해 안드로이드UI핸들러인 handler를 통해 값을 전달한다.     public void loginProcess() {           final ResponseHandler<String> responseHandler=                new ResponseHandler<String>() {                   @Override                  public String handleResponse(HttpResponse response)                                                   throws ClientProtocolException, IOException {                          String result=null;                          HttpEntity entity=response.getEntity();                           **result=parsingData(entity.getContent());**                          Message message=handler.obtainMessage();                          Bundle bundle=new Bundle();                          if ( result.equals("success") )                                        bundle.putString("RESULT", "success");                          else                                       bundle.putString("RESULT", "failed");                          message.setData(bundle);                          **handler.sendMessage(message);**                          return result;                  }     };         // 로그인이 처리되고 있다는 다이얼로그를 화면에 표시한다.      pDialog=ProgressDialog.show(this, "", "로그인 처리중....");         // 서버에 HTTP 처리 요청은 새로운 스레드를 생성하여 비동기식으로 처리하는것이 효율적이다.      new Thread() {             @Override             public void run() {                  String url = "<http://192.168.10.2:8080/login.jsp>";                  HttpClient http = new DefaultHttpClient();                  try {                       // 서버에 전달할 파라메터 세팅                         ArrayList<NameValuePair> nameValuePairs =                                                                new ArrayList<NameValuePair>();                      nameValuePairs.add(new BasicNameValuePair("id", id.getText().toString()));                      nameValuePairs.add(new BasicNameValuePair("passwd", passwd.getText().toString()));                     //     응답시간이 5초가 넘으면 timeout 처리하려면 아래 코드의 커맨트를 풀고 실행한다.                    //     HttpParams params = http.getParams();                    //     HttpConnectionParams.setConnectionTimeout(params, 5000);                    //     HttpConnectionParams.setSoTimeout(params, 5000);    **// HTTP를 통해 서버에 요청을 전달한다.**    // 요청에 대한결과는 responseHandler의 handleResponse()메서드가 호출되어 처리한다.                    // 서버에 전달되는 파라메터값을 인코딩하기위해 UrlEncodedFormEntity() 메서드를 사용한다.                    HttpPost httpPost = new HttpPost(url);                   UrlEncodedFormEntity entityRequest =                                         new UrlEncodedFormEntity(nameValuePairs, "UTF-8");                   httpPost.setEntity(entityRequest);                   **http.execute(httpPost,responseHandler);**                  }catch(Exception e){e.printStackTrace();}            }       }.start();    //스레드를 실행시킨다.  }  } |

LoginActivity를 실행하기 위해 테스트서버를 만들었고, 테스트서버에는  로그인을 처리하기위한 login.jsp, 결과로 반환될 success.xml, failed.xml,을 다음과 같이 만들어 테스트 하였다. ㅎㅎ

|  |
| --- |
| **[login.jsp]**    <%@ page language="java" contentType="text/html; charset=UTF-8"     pageEncoding="UTF-8"%> <%@ taglib prefix="c" uri="<http://java.sun.com/jsp/jstl/core>" %> <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "<http://www.w3.org/TR/html4/loose.dtd>"> <html> <head> <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"> <title>Insert title here</title> </head> <body>    <% System.out.println("login.jsp가 호출됨");  %>    <c:if test="${param.id=='kim'}" >        <c:if test="${param.passwd=='111'}" >             <jsp:forward page="success.xml"/>        </c:if>    </c:if>    <jsp:forward page="failed.xml"/> </body> </html>  **[success.xml]**  <?xml version="1.0" encoding="UTF-8"?> <result>success</result>    **[failed.xml]**  <?xml version="1.0" encoding="UTF-8"?> <result>failed</result> |

|  |  |  |
| --- | --- | --- |
|  | |  | | --- | |  |   [**A simple RSS reader, using Android's org.xml.sax package.**](http://android-er.blogspot.com/2010/04/simple-rss-reader-using-androids.html)   "[Read XML Resources in Android, using XmlResourceParser: XML parsing interface](http://android-er.blogspot.com/2010/04/read-xml-resources-in-android-using.html)",  안드로이드 어플리케이션에서  안드로이드 SAX API를 이용해서 외부의 XML파일을 읽어서 처리한다.    XML소스는  "<http://feeds.feedburner.com/Android-er?format=xml>"  블로그의 RSS feed 이다.  [SAX](http://www.saxproject.org/) 는 원래는 자바 온리 API였던 XML을 위한  Simple API  이다.  현재 SAX 2.0.1 버젼을 사용한다. [org.xml.sax](http://developer.android.com/reference/org/xml/sax/package-summary.html) 은 안드로이드 패키지에서 제공하는 코어 SAX API이다. 좀더 간단하게 만들기위해 title 테그아래의 컨텐츠만 읽어스트링으로 처리한다.  [http://2.bp.blogspot.com/_C5a2qH8Y_jk/S9CKvRG6znI/AAAAAAAAAag/m5DQ6UoSlI8/s400/AndroidRssReader_01.png](http://2.bp.blogspot.com/_C5a2qH8Y_jk/S9CKvRG6znI/AAAAAAAAAag/m5DQ6UoSlI8/s1600/AndroidRssReader_01.png)  안드로이드 어플리케이션에  "android.permission.INTERNET"  퍼미션이 설정되어 있어야 한다.    **AndroidManifest.xml** <?xml version="1.0" encoding="utf-8"?> <manifest xmlns:android="[http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android)           package="com.exercise.AndroidRssReader"           android:versionCode="1"           android:versionName="1.0"> <application android:icon="@drawable/icon" android:label="@string/app\_name"> <activity android:name=".AndroidRssReader"           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> <uses-sdk android:minSdkVersion="4" /> <uses-permission android:name="android.permission.INTERNET" /> </manifest>  **Modify main.xml to add a TextView to display the result.** <?xml version="1.0" encoding="utf-8"?> <LinearLayout xmlns:android="[http://schemas.android.com/apk/res/android"](http://schemas.android.com/apk/res/android)               android:orientation="vertical"               android:layout\_width="fill\_parent"               android:layout\_height="fill\_parent" > <TextView         android:layout\_width="fill\_parent"         android:layout\_height="wrap\_content"         android:text="@string/hello" /> <ScrollView         android:layout\_width="fill\_parent"         android:layout\_height="wrap\_content">    <TextView            android:layout\_width="fill\_parent"            android:layout\_height="wrap\_content"            android:id="@+id/result" /> </ScrollView> </LinearLayout>   **Java source code.** package com.exercise.AndroidRssReader;  import java.io.IOException; import java.net.MalformedURLException; import java.net.URL;  import javax.xml.parsers.ParserConfigurationException; import javax.xml.parsers.SAXParser; import javax.xml.parsers.SAXParserFactory;  import org.xml.sax.Attributes; import org.xml.sax.InputSource; import org.xml.sax.SAXException; import org.xml.sax.XMLReader; import org.xml.sax.helpers.DefaultHandler;  import android.app.Activity; import android.os.Bundle; import android.widget.TextView;  public class AndroidRssReader extends Activity {  String streamTitle = "";  /\*\* Called when the activity is first created. \*/ @Override public void onCreate(Bundle savedInstanceState) {     super.onCreate(savedInstanceState);     setContentView(R.layout.main);      TextView result = (TextView)findViewById(R.id.result);      try {         URL **rssUrl** = new URL(<http://feeds.feedburner.com/Android-er?format=xml>);           SAXParserFactory mySAXParserFactory **= SAXParserFactory.newInstance();**         SAXParser **mySAXParser** = **mySAXParserFactory.newSAXParser();**         XMLReader **myXMLReader** = **mySAXParser.getXMLReader();**         RSSHandler **myRSSHandler** = **new RSSHandler();**         **myXMLReader.setContentHandler(myRSSHandler);**         **InputSource myInputSource = new InputSource(rssUrl.openStream());**         **myXMLReader.parse(myInputSource);**          result.setText(streamTitle);       } catch (MalformedURLException e) {           // TODO Auto-generated catch block           e.printStackTrace();           result.setText("Cannot connect RSS!");      } catch (ParserConfigurationException e) {          // TODO Auto-generated catch block          e.printStackTrace();          result.setText("Cannot connect RSS!");      } catch (SAXException e) {           // TODO Auto-generated catch block          e.printStackTrace();          result.setText("Cannot connect RSS!");      } catch (IOException e) {           // TODO Auto-generated catch block          e.printStackTrace();          result.setText("Cannot connect RSS!");    }   }  **private class RSSHandler extends DefaultHandler** {     final int stateUnknown = 0;     final int stateTitle = 1;     int state = stateUnknown;      int numberOfTitle = 0;     String strTitle = "";     String strElement = "";      @Override     public void startDocument() throws SAXException {           // TODO Auto-generated method stub           strTitle = "--- Start Document ---\n";     }     @Override    public void endDocument() throws SAXException {        // TODO Auto-generated method stub        strTitle += "--- End Document ---";        streamTitle = "Number Of Title: " + String.valueOf(numberOfTitle) + "\n"                       + strTitle;    }     @Override    public void startElement(String uri, String localName, String qName,                              Attributes attributes) throws SAXException {        // TODO Auto-generated method stub     if (localName.equalsIgnoreCase("title"))     {           state = stateTitle;           strElement = "Title: ";           numberOfTitle++;     } else {           state = stateUnknown;    }   }    @Override   public void endElement(String uri, String localName, String qName)               throws SAXException {         // TODO Auto-generated method stub        if (localName.equalsIgnoreCase("title"))        {             strTitle += strElement + "\n";        }       state = stateUnknown;    }    @Override   public void characters(char[] ch, int start, int length)               throws SAXException {         // TODO Auto-generated method stub        String strCharacters = new String(ch, start, length);        if (state == stateTitle)        {              strElement += strCharacters;         }   }   } }  //////////////////////////////////////////////////////////////////// WebView WebView는 웹브라우저 액티비티를 만들수 있도록 해준다.    STEP1. layout file을 다음과 같이 생성한다.  <?xml version="1.0" encoding="utf-8"?> <WebView          xmlns:android="<http://schemas.android.com/apk/res/android>"         android:id="@+id/webview"         android:layout\_width="fill\_parent"         android:layout\_height="fill\_parent"/>    STEP2. 자 이제 WebView를 사용하는 Activity를 만들어 볼까? 우선 onCreate()에서 WebView를 건저와야지.    webview = (WebView) findViewById(R.id.webview); webview.getSettings().setJavaScriptEnabled(true); webview.loadUrl("[http://www.google.com](http://www.google.com/)");    STEP3. 그런다음 WebView가 실행되게 할라면 퍼미션이 있어야 하겠지?             manifest.xml 파일에 다음과 같이 퍼미션을 설정해    <uses-permission android:name="android.permission.INTERNET" />    STEP4. 내가 만든 브라우저에서 앞으로, 뒤로, 확대, 축소 이런 기능이 가능하게 하려면 WebViewClient 객체를 구현해서         만들면 OK          private class HelloWebViewClient extends WebViewClient {        @Override        public boolean shouldOverrideUrlLoading(WebView view, String url){            view.loadUrl(url);            return true;        }              @Override        public void onPageFinished(WebView view, String url){        }           @Override         public boolean shouldOverrideKeyEvent (WebView view, KeyEvent event) {            int keyCode = event.getKeyCode();            if ((keyCode == KeyEvent.KEYCODE\_DPAD\_LEFT) && webview.canGoBack()) {                webview.goBack();                return true;            }else if ((keyCode == KeyEvent.KEYCODE\_DPAD\_RIGHT) && webview.canGoForward()) {                webview.goForward();                return true;            }            return false;       }   }    STEP5.  이렇게 만든 WebViewClient를 사용하려면 onCreate()에서 다음과 같이 코딩한다.  **webview.setWebViewClient(new HelloWebViewClient ());**      STEP6.  Activity를 실행하면 다음과 같은 결과가 짜잔~~  http://developer.android.com/guide/tutorials/views/images/hello-webview.png  ////////////////////////////////////////////////////////////////////  [[안드로이드예제] HttpClient+WebView를 이용한 웹사이트 정보 긁어와서 뿌리기](http://blog.naver.com/q1q3q5/10099522861)  package kim.project;  // import 생략  public class HTTPTest extends Activity {     String bundleResult;     ProgressDialog progressDialog;     String TAG="HTTPTest";     TextView text;     WebView webview;    //HttpResponse 결과를 받아서 화면에 WebView로 뿌리는 UI핸들러    private final Handler handler = new Handler() {           @Override          public void handleMessage(final Message msg) {              progressDialog.dismiss();              bundleResult = msg.getData().getString("RESPONSE");             try {              **webview.loadDataWithBaseURL("",bundleResult, "text/html", "utf-8", null);**             }catch(Exception e){}             }      };          @Override   protected void onCreate(Bundle savedInstanceState) {    // TODO Auto-generated method stub    super.onCreate(savedInstanceState);        setContentView(R.layout.web\_layout);    webview=(WebView)findViewById(R.id.webview);    webview.getSettings().setDefaultTextEncodingName("euc-kr");    webview.getSettings().setJavaScriptEnabled(true);    performRequest();       }   @Override      public void onPause() {          super.onPause();          if (this.progressDialog != null && this.progressDialog.isShowing()) {              this.progressDialog.dismiss();          }      }             private void performRequest() {          //HttpResponse를 처리할 핸들러객체 생성           final ResponseHandler<String> responseHandler =             new ResponseHandler<String>() {                public String handleResponse(HttpResponse response) {                   StatusLine status = response.getStatusLine();                   String result = "검색결과가 없습니다.";                   try {                   if ( status.getStatusCode() == 200) {                                       HttpEntity entity = response.getEntity();                                                          result = parsingData(entity.getContent());                         }                       Log.i(TAG,result);                       Message message = handler.obtainMessage();                       Bundle bundle = new Bundle();                       bundle.putString("RESPONSE", result);                       message.setData(bundle);                       handler.sendMessage(message);                   } catch (IOException e) {                       Log.e(TAG,  e.toString());                   }                   return result;               }           };            this.progressDialog =             ProgressDialog.show(this, "working . . .", "performing HTTP request");              // 스레드를 하나 띄워서 HttpRequest를 전송            new Thread() {                @Override               public void run() {                   try {                       **DefaultHttpClient client = new DefaultHttpClient();**                       String url="[http://www.chterminal.co.kr/sub3/sub3\_1.asp?query="+URLEncoder.encode("서울","EUC-KR](http://www.chterminal.co.kr/sub3/sub3_1.asp?query=%22+URLEncoder.encode(%22서울%22,%22EUC-KR)");                       Log.i("Query",url);                       **HttpGet httpMethod = new HttpGet(url);** **httpMethod.setHeader("Content-Type",**  **"application/x-www-form- urlencoded");                   client.execute(httpMethod, responseHandler);**                   } catch (ClientProtocolException e) {                       Log.e(TAG,  e.toString());                   } catch (IOException e) {                       Log.e(TAG,  e.toString());                   }               }           }.start();       }         // 요기는 무식한 방법으로 필요한 HTML 코드 부분만 뽑아온 코드     // Response내용중 두번째 이후 <table>에 들어있는 내용만 뽑아와서     // 화면에 뿌려주기위한 필터링작업 ㅋㅋ      public String  parsingData(InputStream input){           StringBuffer sb=new StringBuffer();             try {           BufferedReader reader=            new BufferedReader(new InputStreamReader(input,"euc-kr"));           String line=null;               boolean flag=false;           int count=0;           while ( (line=reader.readLine()) != null ) {                       Log.i("input....",line);                      if ( line.contains("<table ")) {             flag=true;             count++;            } else if ( line.contains("</table ")) {                         flag=false;             if (count > 0) sb.append(line);            }            if (count > 0 && flag == true) sb.append(line);                               }            }catch(Exception e){e.printStackTrace();}              return sb.toString();    }  }      >>  요거는 오랜삽질 끝에 나에게 답을 준 고마운 게시물.  webview.loadData()가지고 안만해도 한글문제가 해결이 안되더니만 이분께서 먼저 삽질하시고 친절하게 해결책을 주셨구먼  역시 지식은 나눠야해.  감사해요^^  출처: <http://ecolemodev.springnote.com/pages/4742165>  WebView에서 컨텐트를 로드하는 방법은 다음 세 가지가 있다.   * loadUrl * loadData * loadDataWithBaseURL   한글 인코딩이 문제가 되는 건 html을 코드에서 생성해서 넘겨줄 때다. 그냥 API를 보면 loadData를 쓰면 된다고 생각하기 쉬운데, 사실 loadData의 구현을 보면 loadUrl을 사용하되 data:라는 scheme을 사용하게 되어 있어서 loadUrl에 더 가깝다. 그래서 실제로 loadData의 content에 html을 줄 경우 이걸 URL로 인식하기 때문에 URLEncoder로 encode해주지 않으면 한글이 깨진다. 또, 인코딩을 해주면 html의 태그도 같이 encode되서 html이 안 나온다. 그래서, 사실은 loadData는 거의 쓸모가 없다.  대신 loadDataWithBaseURL 쓰면 된다. 다음과 같이 쓰면 문제 없이 한글도, html도 나온다.  webView.loadDataWithBaseURL(url, htmlContent, "text/html", "UTF-8", null);  url에는 임의로 아무 거나 넣어주면 된다. browser의 back/forward 기능에서 쓰기 위함이라고 하니 unique하면 좋을 것이다. url에 null을 넣으면 about:blank로 연결되기 때문에 아무 것도 안 나올 수도 있다. 그냥 의미 없지만 unique한 스트링을 아무렇게나 넘겨주면 된다.  /////////////////////////////////////////////////////////////////////////////  package kim.project;  // import 생략  public class WebViewTest3 extends Activity {         String bundleResult;    ProgressDialog progressDialog;    String TAG="WebViewTest3";    static final String URL=[http://www.naver.com"](http://www.naver.com/)     TextView text;  // HTTP 요청결과를 message로 받아서 화면에 뿌리는 작업을 수행  private final Handler handler = new Handler() {          @Override         public void handleMessage(final Message msg) {             progressDialog.dismiss();             bundleResult = msg.getData().getString("RESPONSE");              text.setText(bundleResilt);     // 화면의 TextView에 읽어온 html 코드를 그대로 출력         }     };        @Override  protected void onCreate(Bundle savedInstanceState) {      // TODO Auto-generated method stub       super.onCreate(savedInstanceState);       setContentView(R.layout.web\_layout);        text=(TextView)findViewById(R.id.text01);        performRequest();     }  @Override     public void onPause() {         super.onPause();         if (this.progressDialog != null && this.progressDialog.isShowing()) {             this.progressDialog.dismiss();         }     }           private void performRequest() {           // HTTP 요청에 대한 결과를 받아서 처리하는 response 핸들러           final ResponseHandler<String> responseHandler =            new ResponseHandler<String>() {               public String handleResponse(HttpResponse response) {                  StatusLine status = response.getStatusLine();                  HttpEntity entity = response.getEntity();                  String result = null;                  try {                      result = parsingData(entity.getContent());                                      Log.i(TAG,result);                      Message message = handler.obtainMessage();  // handler의 Message객체를 얻어온다.                      Bundle bundle = new Bundle();                      bundle.putString("RESPONSE", result);                      message.setData(bundle);                      handler.sendMessage(message);   // 응답받은 데이터를 메시지에 세팅하여 handler에게 전달                  } catch (IOException e) {                      Log.e(TAG,  e.toString());                  }                  return result;              }          };           this.progressDialog = ProgressDialog.show(this, "working . . .", "performing HTTP request");           // do the HTTP dance in a separate thread (the responseHandler will fire when complete)          new Thread() {               @Override              public void run() {                  try {                      DefaultHttpClient client = new DefaultHttpClient();                      HttpGet httpMethod = new HttpGet(URL);   // GET방식으로 요청                      //httpMethod.setHeader("Content-Type", "application/x-www-form- urlencoded");                      client.execute(httpMethod, responseHandler);  // 요청 실행, 실행결과 Response는 responseHandler가 처리                  } catch (ClientProtocolException e) {                      Log.e(TAG,  e.toString());                  } catch (IOException e) {                      Log.e(TAG,  e.toString());                  }              }          }.start();      }        public String  parsingData(InputStream input){  //    parsing이 필요하면 여기서 아래 코드 대신 parsing코드를 대신 작성함.  //           StringBuffer sb=new StringBuffer();  //            try {            //             XmlPullParserFactory factory = XmlPullParserFactory.newInstance();  //             factory.setNamespaceAware(true);  //             XmlPullParser parser = factory.newPullParser();  //             parser.setInput(input, "euc-kr");    // InputStream을 xml parser의 입력으로  //                   //              //             while (parser.next()  != XmlPullParser.END\_DOCUMENT) {      //                   String name=parser.getName(); //                   if( name!= null && name.equals("td")) { //                        sb.append(parser.getText()+"\n");   // td 엘리먼트의 값을 버퍼에 저장하는 예 //                   } //               } //            }catch(Exception e){e.printStackTrace();}          BufferedReader reader=new BufferedReader(new InputStreamReader(input,"utf-8"));          StringBuffer sb=new StringBuffer()          String line=null;          while( (line=reader.readLine()) != null ) {                      sb.append(line);          }          return sb.toString();     } }  **////////////////////////////////////////////////////////////////////////////////////////////////////**  package kim.project;  import java.io.BufferedReader; import java.io.IOException; import java.io.InputStream; import java.io.InputStreamReader; import java.net.URLEncoder;  import org.apache.http.HttpEntity; import org.apache.http.HttpResponse; import org.apache.http.StatusLine; import org.apache.http.client.ClientProtocolException; import org.apache.http.client.ResponseHandler; import org.apache.http.client.methods.HttpGet; import org.apache.http.impl.client.DefaultHttpClient;  import android.app.Activity; import android.app.ProgressDialog; import android.os.Bundle; import android.os.Handler; import android.os.Message; import android.util.Log; import android.widget.TextView;  public class HTTPTest extends Activity {     String bundleResult;     ProgressDialog progressDialog;     String TAG="HTTPTest";     TextView text;        private final Handler handler = new Handler() {           @Override          public void handleMessage(final Message msg) {              progressDialog.dismiss();              bundleResult = msg.getData().getString("RESPONSE");              text.setText(bundleResult);          }      };          @Override   protected void onCreate(Bundle savedInstanceState) {    // TODO Auto-generated method stub    super.onCreate(savedInstanceState);    setContentView(R.layout.http\_layout);    text=(TextView)findViewById(R.id.text01);    performRequest();       }   @Override      public void onPause() {          super.onPause();          if (this.progressDialog != null && this.progressDialog.isShowing()) {              this.progressDialog.dismiss();          }      }             private void performRequest() {           final ResponseHandler<String> responseHandler =             new ResponseHandler<String>() {                public String handleResponse(HttpResponse response) {                   StatusLine status = response.getStatusLine();                   String result = "검색결과가 없습니다.";                   try {                   if ( status.getStatusCode() == 200) {                                       HttpEntity entity = response.getEntity();                                                          result = parsingData(entity.getContent());                         }                       Log.i(TAG,result);                       Message message = handler.obtainMessage();                       Bundle bundle = new Bundle();                       bundle.putString("RESPONSE", result);                       message.setData(bundle);                       handler.sendMessage(message);                   } catch (IOException e) {                       Log.e(TAG,  e.toString());                   }                   return result;               }           };            this.progressDialog =             ProgressDialog.show(this, "working . . .", "performing HTTP request");            // do the HTTP dance in a separate thread (the responseHandler will fire when complete)           new Thread() {                @Override               public void run() {                   try {                       DefaultHttpClient client = new DefaultHttpClient();                       String url="[http://www.chterminal.co.kr/sub3/index.asp?query="+URLEncoder.encode("서울","EUC-KR](http://www.chterminal.co.kr/sub3/index.asp?query=%22+URLEncoder.encode(%22서울%22,%22EUC-KR)");                       Log.i("Query",url);                       HttpGet httpMethod = new HttpGet(url);                       httpMethod.setHeader("Content-Type", "application/x-www-form- urlencoded");                       client.execute(httpMethod, responseHandler);                   } catch (ClientProtocolException e) {                       Log.e(TAG,  e.toString());                   } catch (IOException e) {                       Log.e(TAG,  e.toString());                   }               }           }.start();       }          public String  parsingData(InputStream input){           StringBuffer sb=new StringBuffer();             try {           BufferedReader reader=            new BufferedReader(new InputStreamReader(input,"euc-kr"));           String line=null;                  while ( (line=reader.readLine()) != null ) {            sb.append(line); //           if ( line.contains("table")) { //            sb.append(line); //            while ( ! (line=reader.readLine()).contains("table")) { //             sb.append(line); //            } //           }                     }           }catch(Exception e){e.printStackTrace();} //             try { //               //              XmlPullParserFactory factory = XmlPullParserFactory.newInstance();  //              factory.setNamespaceAware(true);  //              XmlPullParser parser = factory.newPullParser();  //              parser.setInput(input, "euc-kr"); //                    //               //              while (parser.next()  != XmlPullParser.END\_DOCUMENT) {      //               String name=parser.getName(); //               if( name!= null && name.equals("td")) { //                Log.i(TAG,name); //       sb.append(parser.nextText()+"\n"); //      } //                } //             }catch(Exception e){e.printStackTrace();}              return sb.toString();     }    }  /////////////////////////////////////////////////////////////////////////  **package** com.android.hello;    **import** com.google.android.maps.MapActivity;  **import** com.google.android.maps.MapView;  **import** com.google.android.maps.GeoPoint;  **import** com.google.android.maps.MyLocationOverlay;    **import** android.os.Bundle;      **public** **class** HelloAndroid **extends** MapActivity {     /\*\* Called when the activity is first created. \*/  **private** MapView mapView;  **private** MyLocationOverlay myLocationOverlay;  @Override     **public** **void** onCreate(Bundle icicle) {        **super**.onCreate(icicle);  setContentView(R.layout.*main*);  mapView = (MapView) findViewById(R.id.*simpleGM\_map*);    myLocationOverlay = **new** MyLocationOverlay(**this**, mapView);  **if**(myLocationOverlay.isMyLocationEnabled()) {  myLocationOverlay.runOnFirstFix(**new** Runnable(){**public** **void** run(){  mapView.getController().animateTo(myLocationOverlay.getMyLocation());  }});  mapView.getOverlays().add(myLocationOverlay);  mapView.getController().setZoom(5);  mapView.setClickable(**true**);  mapView.setEnabled(**true**);  mapView.setSatellite(**true**);  } **else** {  GeoPoint point = **new** GeoPoint(37497618,127031039);  mapView.getController().animateTo(point);  mapView.getController().setZoom(5);  mapView.setClickable(**true**);  mapView.setEnabled(**true**);  mapView.setSatellite(**true**);  }    }    @Override  **protected** **void** onResume(){  **super**.onResume();  myLocationOverlay.enableMyLocation();  }    @Override  **protected** **void** onStop(){  myLocationOverlay.disableMyLocation();  **super**.onStop();  }    @Override  **protected** **boolean** isRouteDisplayed() {  // **TODO** Auto-generated method stub  **return** **false**;  }  }  **////////////////////////////////////////////////////////////////////////////////////////////////////////**   1. **try** { 2. //다운로드 받을 URL에 대한 객체를 생성한다. 3. // 4. URL url = **new** URL("[http://somewhere.com/some/webhosted/file"](http://somewhere.com/some/webhosted/file)); 6. // URL로 부터 connection 객체를 를 생성한다. 7. HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection(); 9. //생성된 커넥션에 대해 추가적인 설정을 지정한다. 여기서는 get방식의 요청을 설정 10. urlConnection.setRequestMethod("GET"); 11. urlConnection.setDoOutput(**true**); 13. //웹서버에 연결한다. 14. urlConnection.connect(); 16. //파일을 저장하기 위한 경로를 지정한다. 17. //이 예제에서는 sdcard의 root 디렉토리에 파일을 저장한다. 18. // 19. File SDCardRoot = Environment.getExternalStorageDirectory(); 20. //SDCard root 디렉토리에 somefile.txt 파일에 파일을 저장하기 위해 21. //파일 객체를 생성한다. 22. File file = **new** File(SDCardRoot,"somefile.ext"); 24. //파일을 오픈한다. 25. FileOutputStream fileOutput = **new** FileOutputStream(file); 27. //인터넷으로 부터 데이터를 읽어들이기 위한 입력스트림을 얻어온다. 28. InputStream inputStream = urlConnection.getInputStream(); 30. //파일의 전체 크기를 얻어온다. 31. **int** totalSize = urlConnection.getContentLength(); 32. //다운로드 받을 전체 바이트 크기를 변수에 저장한다. 33. **int** downloadedSize = 0; 35. //버퍼를 생성한다. 36. **byte**[] buffer = **new** **byte**[1024]; 37. **int** bufferLength = 0; //임시로 사용할 버퍼의 크기 지정 39. //입력버퍼로 부터 데이터를 읽어서 내용을 파일에 쓴다. 40. **while** ( (bufferLength = inputStream.read(buffer)) > 0 ) { 41. //버퍼에 읽어들인 데이터를 파일에 쓴다. 42. fileOutput.write(buffer, 0, bufferLength); 43. //다운로드 받은 바이트수를 계산한다. 44. downloadedSize += bufferLength; 45. //progressDialog에 다운로드 받은 바이트수를 표시해 준다.  <-따로 progressDialog스레드를 작성해둘것 46. updateProgress(downloadedSize, totalSize); 48. } 49. //작업이 끝나면 파일을 close하여 저장한다. 50. fileOutput.close(); 52. //예외가 발생한 경우 다음 catch 루틴을 처리한다. 53. } c**atch** (MalformedURLException e) { 54. e.printStackTrace(); 55. } **catch** (IOException e) { 56. e.printStackTrace(); 57. }   **//////////////////////////////////////////////////////////////////////////////////////////////////////**  HttpPost 요청처리하기   1. **public** **void** postData() { 2. // 새로운 HttpClient와 Post헤더를 생성한다. 3. HttpClient httpclient = **new** DefaultHttpClient(); 4. HttpPost httppost = **new** HttpPost("[http://www.test.com/script.php"](http://www.test.com/script.php)); 6. **try** { 7. // Body에 담아가야할 데이터를 만들고 body에 담기 8. List<NameValuePair> nameValuePairs = **new** ArrayList<NameValuePair>(2); 9. nameValuePairs.add(**new** BasicNameValuePair("id", "guest")); 10. nameValuePairs.add(**new** BasicNameValuePair("passwd", "1234")); 11. httppost.setEntity(**new** UrlEncodedFormEntity(nameValuePairs)); 13. // Http Post 요청을 처리한다. 14. HttpResponse response = httpclient.execute(httppost); 16. } **catch** (ClientProtocolException e) { 17. // TODO Auto-generated catch block 18. } **catch** (IOException e) { 19. // TODO Auto-generated catch block 20. } 21. }   ///////////////////////////////////////////////////////////////////////////  **>> HttpGet 방식예**    try {     HttpClient client = new DefaultHttpClient();     String getURL = "[http://www.sun.com/page.jsp";](http://www.sun.com/page.jsp)     HttpGet get = new HttpGet(getURL);     HttpResponse responseGet = client.execute(get);     HttpEntity resEntityGet = responseGet.getEntity();     if (resEntityGet != null) {           //do something with the response           Log.w("RESPONSE",EntityUtils.toString(resEntityGet));     }  } catch (Exception e) {     e.printStackTrace();  }        **>> HttpPost 방식예**    try {      HttpClient client = new DefaultHttpClient();      String postURL = "[http://www.sun.com/page.jsp";](http://www.sun.com/page.jsp)      HttpPost post = new HttpPost(postURL);      List<NAMEVALUEPAIR> params = new      ArrayList<NAMEVALUEPAIR>();      params.add(new BasicNameValuePair("user", "guest"));      params.add(new BasicNameValuePair("pass", "1234"));      UrlEncodedFormEntity ent = new                    UrlEncodedFormEntity(params,HTTP.UTF\_8);      post.setEntity(ent);      HttpResponse responsePOST = client.execute(post);      HttpEntity resEntity = responsePOST.getEntity();       if (resEntity != null) {             Log.w("RESPONSE",EntityUtils.toString(resEntity));       }  } catch (Exception e) {       e.printStackTrace();  }      **>> Multipart Post 방식예**    File file = new File("files/sample.txt");  try {        HttpClient client = new DefaultHttpClient();        String postURL = "[http://www.sun.com/page.jsp";](http://www.sun.com/page.jsp)        HttpPost post = new HttpPost(postURL);        FileBody bin = new FileBody(file);        MultipartEntity reqEntity = new        MultipartEntity(HttpMultipartMode.BROWSER\_COMPATIBLE);        reqEntity.addPart("myFile", bin);        post.setEntity(reqEntity);        HttpResponse response = client.execute(post);        HttpEntity resEntity = response.getEntity();        if (resEntity != null) {             Log.w("RESPONSE",EntityUtils.toString(resEntity));        }  } catch (Exception e) {        e.printStackTrace();  }  **////////////////////////////////////////////////////////////////////////////////////////////////////////////**  **<프로그램의 구성>**      프로그램은 코드와 리소스로 구성한다. 여러가지 이유로 코드와 데이터는 가급적 분리하는 것이 좋다.      코드는 일정한 문법에 따라 자바 소스 파일에 작성되며, 프로그램의 흐름과 동작을 정의한다.      리소스는 동작과 직접적인 상관이 없으며, 주로 프로그램의 외형 장식을 정의한다.      리소스와 코드를 분리하면 좋은 이유는 코드는 프로그래머가 작성하고, 리소스는 디자이너가 작성함으로써 병행 작업이 가능하다.      한마디로 개발 속도도 높이고, 관리도 용이하다.      **<리소스>**      리소스란 응용 프로그램 실행에 필요한 일체의 데이터를 합쳐서 '리소스'라고 한다.      리소스의 예로는 문자열이나, 이미지, 사운드 같은 것들이 있다.      리소스는 환경이나 조건에 따라, 통째로 교체할 수 있으므로 장비 호환성 확보에 유리하며 언어에 따른 지역화도 간단해진다.      뭐든 분리해 놓으면 관리하기 편해지며, 합치는 과정에서 다양한 기교를 부릴 수 있어 응용의 여지가 생긴다.        안드로이드는 프로젝트도 현대적 추세에 따라 res폴더와 assets폴더로 좀 더 세분화하여 관리한다.      이 둘의 차이를 설명하자면 이렇다.      (1) 리소스에 비해 에셋은 동영상 같은 큰 데이터를 의미한다.    (2) 리소스는 컴파일되지만, 에셋은 원본 그대로 저장된다.    (3) 리소스는 빈번히 사용되는데 비해, 에셋은 상대적으로 덜 사용된다.      한마디로 assets은 대용량 파일을 넣고, res는 저용랑 파일을 넣는다.      **리소스의 종류**      리소스도 종류가 여러 개 있다. 리소스 폴더 이름은 강제적으로 정해져 있으므로, 타입에 맞는 폴더에 집어넣어야 한다.      각 폴더에 저장되는 리소스 타입은 다음과 같다.    http://cafefiles.naver.net/20101001_82/knossos88_1285919733622Jtsih_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png        그리고 이 value 폴더에는 문자열, 색상, 배열, 크기, 스타일 등 다양한 자료가 들어가는만큼, 리소스 폴더처럼 또 타입이 필요하다.    http://cafefiles.naver.net/20101001_179/knossos88_12859207312657WH56_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      그런데 이는 권장사항이지, 강제는 아니므로 꼭 지키지 않아도 된다.      다만 소스를 여기저기 흩어 놓으면 관리하기 어려우므로, 종류 별로 관리하자는 것이다.        다시 정리하면, 앞의 폴더는 꼭 지켜줘야하며, 리소스는 임의대로 생성해도 되나 가급적 권장 사항에 따르자는 내용이었다.      **aapt**      aapt란 res 하위 폴더에 저장된 리소스를 컴파일하여 패키징하는 것인데, 그냥 이런 것이 있다는 것만 알아두고 넘어가면 된다.      aapt가 하는 일은 (1) 폴더의 이름과 리소스 파일의 확장자에 따라 파일 포맷을 판별하여 미리 정의된 방식으로 리소스를 컴파일하기      (2) res 하위 폴더에 있는 모든 리소스 ID를 검색하여, gen 폴더의 R.java 파일을 만들어 내는 것이다.      한마디로 aapt는 res 안에 모든 리소스를 컴파일해서 넘겨준다. 그리고 R.java 파일을 만든다.      **리소스 사용**      증요한 것은 aapt가 어떤 존재인가는 별로 중요하지 않다.      어차피 컴퓨터에서 해주는 것은 컴퓨터가 해주기 때문이다. (당신이 공학 박사가 생각이 아니라면 말이다)        리소스 코드는 다방면으로 활용된다. 대표적으로 setContentView인데      이 메서드는 본디 액티비티에 채울 뷰를 요구하는데, 매번 생성하기 번거로우니 ID만 받아 내부적으로 직접 전개하도록 설계되었다.      객체 지향 클래스들의 메서드들은 가능한 모든 타입의 인수를 받아들여, 최종적으로 필요한 정보를 내부적 처리를 하도록 되어 있다.        뭐 이모저모 빙빙 돌려 얘기했는데, 간단하다. 이제껏 우리가 했던 것을 체계적으로 얘기하는 것일 뿐.      리소스를 사용하여 전개하는 방법은 크게 2가지다.        첫번째는, 각각의 XML 문서를 코드에서 불러와 참조하는 것이고      두번째는, layout에 있는 XML에 모두 등록해놓고 코드에서 한번에 불러내는 것이다.      이것을 코드로 풀어내면 이렇다.        (1) res 폴더의 하위 폴더에 각각 저장해두고, 코드에서 각각 호출하기           우리가 앞서 보았던 Strings, colors, dimens 등을 각각 폴더에 저장한 후 부르는 것이다.      public class ReadResource extends Activity {     public void onCreate(Bundle savedInstanceState) {       super.onCreate(savedInstanceState);       setContentView(R.layout.widget\_readresource);        Resources res = getResources();          TextView text = (TextView)findViewById(R.id.text);          String str = res.getString(R.string.textstr);       text.setText(str);       //text.setText(R.string.textstr); // (위의 2줄을) 한 줄로 표현하면 이렇다는 뜻이다          int textcolor = res.getColor(R.color.textcolor);       text.setTextColor(textcolor);       //text.setTextColor(R.color.textcolor);          float textsize = res.getDimension(R.dimen.textsize);       text.setTextSize(textsize);       //text.setTextSize(R.dimen.textsize);     }   }        (2) 레이아웃 XML 문서에 등록한 후, 한 번에 호출하기      <?xml version="1.0" encoding="utf-8"?>   <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"     android:orientation="vertical"     android:layout\_width="fill\_parent"     android:layout\_height="fill\_parent"     android:background="#e0e0e0"     >   <TextView     android:layout\_width="wrap\_content"     android:layout\_height="wrap\_content"     android:text="@string/textstr"     android:textSize="@dimen/textsize"     android:textColor="@color/textcolor"     />   </LinearLayout>      이렇게 레이아웃 XML 문서에 모두 등록을 해놓고      public class ReadResource2 extends Activity {     public void onCreate(Bundle savedInstanceState) {       super.onCreate(savedInstanceState);       setContentView(R.layout.widget\_readresource2);     }   }      이처럼 한꺼번에 호출하면 된다.        그런데, 이 리소스 ID를 참조하는 방법이 은근히 헷갈린다. 그래서 이 참에 확실히 정리하고 가자.    http://cafefiles.naver.net/20101001_208/knossos88_1285935775678GMV5E_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png        전혀 새롭지 않은 내용이다. 그동안 했던 아이디 참조를 정리했을 뿐이다.        결국 이번 단원에서 배운 내용은 리소스를 참조하는 방법을 배운 것이다. (정해진 참조폴더를 지켜야 한다는 것을 잊지마라)      간단하게 문자열이나 색상만을 나타냈지만, 이외에도 배열, 애니메이션, 사운드, 동영상, 도형 등 아주 많은 리소스를 정의할 수 있다.      나머지 리소스는 다시 소개할 예정이니 걱정하지 말자. 방법은 여기서 소개한 것과 별반 다르지 않다.        **<스타일과 테마>**      스타일과 테마는 여러가지 속성 값의 집합을 뜻하는 것이다.      이 둘의 차이는 간단하다. 스타일은 뷰 단위, 테마는 액티비티이다.      왠지 이름만 들어도 감이 오지 않는가? 스타일은 사람들처럼 개개인의 스타일, 테마는 스케일이 큰 극장처럼 말이다.        스타일은 아까 말했던대로 속성의 집합이다. 다양한 속성을 지정한 후에 가져다 사용만 하면 일괄적인 모습을 갖게 된다.      무슨 말인가는 직접 해보면 더 빠를 것이다. res/values 폴더에 styles.xml 파일을 생성하고 다음 문서를 작성한다.    **styles.xml**      <?xml version="1.0" encoding="utf-8"?>   <resources>     **<style name="red15">**       **<item name="android:textColor">#ff0000</item>**       **<item name="android:textSize">15pt</item>**     **</style>**     **<style name="yellow15italic" parent="@style/red15">**       **<item name="android:textColor">#ffff00</item>**       **<item name="android:textStyle">italic</item>**     **</style>**     **<style name="mytheme">**               **<item name="android:windowNoTitle">true</item>**     **</style>**   </resources>      스타일을 지정하는 속성은 **<style name = " ">** 이게 끝이다. 간단하지 않은가?      그 속에 item name=" " 해놓고 그 안에 우리가 여태 했던 속성을 집어넣으면 된다.    **parent**는 말 그대로 상속받았다는 얘기다. 상속 받지 않을 때는 생략하면 된다.      이제 이 스타일이 적용되는지 테스트 해볼 차례다. 코드에는 작성할 것이 없다. 불러오기만 하면 된다.      **Widget/layout/styletest.xml**      <?xml version="1.0" encoding="utf-8"?>   <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"     android:orientation="vertical"     android:layout\_width="fill\_parent"     android:layout\_height="fill\_parent"     android:background="#808080"     >   <TextView       android:layout\_width="fill\_parent"      android:layout\_height="wrap\_content"      android:text="green"     android:textColor="#00ff00"     />   <TextView       android:layout\_width="fill\_parent"      android:layout\_height="wrap\_content"      android:text="red 15 point"     **style="@style/red15"**     />   <TextView       android:layout\_width="fill\_parent"      android:layout\_height="wrap\_content"      android:text="yellow 15 point"     **style="@style/yellow15italic"**     />   </LinearLayout>      http://cafefiles.naver.net/20101003_48/knossos88_1286105664843SI1l6_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png        리니어 레이아웃 안에 세 개의 텍스트를 배치하고, 스타일 리소스를 적용해 보았다.      위젯에 style 속성에 "@style/스타일명" 형식으로 지정한다.      안드로이드에 미리 정의된 스타일이 아니라, 응용 프로그램이 정의한 스타일이므로 android: 네임스페이스는 붙이지 말아야 한다.        물론 스타일을 사용하지 않고, 개별 속성을 일일이 지정해도 똑같다.      그러나 스타일을 적용해 놓으면, 스타일만 수정함으로서 참조하는 모든 위젯의 모양을 한꺼번에 변경할 수 있어 편리하다.      조금 위의 예제인, styles.xml의 textSize를 수정해보자.    **styles.xml**      <?xml version="1.0" encoding="utf-8"?>   <resources>     <style name="red15">       <item name="android:textColor">#ff0000</item>       <item name="android:textSize">**8pt**</item> // **15pt를 8pt로 바꾸어보자.**     </style>     <style name="yellow15italic" parent="@style/red15">       <item name="android:textColor">#ffff00</item>       <item name="android:textStyle">italic</item>     </style>     <style name="mytheme">               <item name="android:windowNoTitle">true</item>     </style>   </resources>    http://cafefiles.naver.net/20101003_223/knossos88_12861060088017honz_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      아까보다 확실히 줄어든 것을 볼 수 있다. 스타일을 상속 받았기 때문에 그대로 적용된다.        이번에는 Manifest 파일을 열어서 다음을 복사해 넣어보자.       <activity android:name="exam.Widget.ThemeTest"       android:theme="@style/mytheme"       android:label="@string/app\_name" />      실행해 보면 테마의 지시대로 액티비티에 타이틀 바가 나타나지 않을 것이다. 액티비티 전체를 이용하고자 할 때 유용하다.    http://cafefiles.naver.net/20101003_52/knossos88_1286110382466TLxrk_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      단순히 manifest.xml에서 지정했다고 되는 것이 아니고, 좀 전에 styles.xml에서 이렇게 지정했기 때문이다.     <style name="mytheme">            <item name="android:windowNoTitle">true</item>  </style>        또 다른 한편으론 이런 것도 이용할 수 있다.      SystemTheme 액티비티를 만들고, 또 다음 소스를 Manifest.xml에 추가한다.    <activity android:name="exam.Widget.SystemTheme"      android:theme="@android:style/Theme.Dialog"      android:label="@string/app\_name" />      대화 상자가 이처럼 화면의 중앙에 나타나고, 뒷부분의 배경은 반투명하게 나타난다.    http://cafefiles.naver.net/20101003_238/knossos88_1286109677966IGT3q_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      **대체 리소스**      안드로이드의 개방 환경만큼이나 수많은 하드웨어 구성이 가득하다.      다양한 사용자 취향에 맞출 수 있다는 것은 장점이지만, 개발자 입장에서는 호환성 확보에 많은 신경을 써야 한다.      각종 하드웨어에 맞추려면 아무래도 하나의 소스를 가지고는 한계가 있다.      안드로이드는 환경에 따라 적절한 리소르를 선택하는 방식으로 이 문제를 해결한다.        개발자가 환경별로 사용할 리소스로 따로 작성해 놓으면 시스템은 실행시 가장 적합한 리소스를 골라 불러온다.      예를 들어, 한글과 영어를 작성해놓으면 장비의 언어 설정에 따라 맞는 리소스를 가져온다.      각 환경에 사용할 대체 리소스를 폴더별로 따로 작성하되, 폴더의 이름에 환경을 의미하는 접미어를 붙여놓는다.    http://cafefiles.naver.net/20101003_60/knossos88_12861083826126HsH1_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      (keysexposed는 하드웨어 key 지원, keyshidden은 있으나 사용은 못하고, keysofr는 SW키보드를 제공한다)        각 환경을 의미하는 접미어는 시스템에 의해 미리 정해져 있으며 접미어를 붙이는 데도 엄격한 규칙이 적용된다.      (1) 여러 개의 접미어를 붙일 때 대시(-)로 구분함, 반드시 도표의 순서대로 작성해야 함    (2) 모든 폴더는 같은 부모에 , 중첩은 불가    (3) 접미는 대소문자 구분. 한국어 레이아웃은 layout-kr이어야 하며, layout=KR은 안된다.    (4) 한 환경에 대해 하나의 접미만 가능. layout-kr-en 따위는 할 수 없다.    (5) 코드나 리소스에 참조할 때는 접미를 붙이지 않는다. 접미는 어디까지나 운영체제가 리소스를 선택할 경우에만 쓴다.        지금까지의 예제는 대체 리소스를 정의하지 않았기 때문에 항상 같은 리소스가 로드되었다.      그래서 이제 예제를 통해 대체 리소스에 대해 알아보려고 한다.      그러나 어떤 리소스를 선택할 것인가는 시스템이 자동으로 결정하며, 사용자가 강제로 선택할 수는 없다.      그래서 한국어 장비에서 영문 리소스를 테스트하기가 쉽지 않다. 언어 설정이나 폴더명을 바꾸는 식으로 변칙을 써야 한다.        해상도나 장비 구성도 바꿀 수 있으나 번거로우므로, 가로·세로 방향별로 리소스를 작성해 보자.      **Widget/layout/landport.xml**      <?xml version="1.0" encoding="utf-8"?>   <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"     android:orientation="vertical"     android:layout\_width="fill\_parent"     android:layout\_height="fill\_parent"     android:gravity="center"     android:background="#e0e0e0"     >   <Button     android:layout\_width="100px"      android:layout\_height="wrap\_content"      android:text="버튼이다"     />   <Button     android:layout\_width="100px"      android:layout\_height="wrap\_content"      android:text="수직이다"     />   </LinearLayout>        이 리소스를 사용할 소스 파일은 다음과 같이 작성한다.      public class LandPort extends Activity {     public void onCreate(Bundle savedInstanceState) {       super.onCreate(savedInstanceState);       setContentView(R.layout.**widget\_landport**);     }   }      이처럼 ID를 읽을 때는 접미를 뺀 폴더명만 적으면 된다. 영문을 읽고 싶다고 R.layout-en.landport이라고 하면 안 된다.      실행해 보면 처음엔 밑의 화면을 얻을 수 있다. Ctrl + 11을 누르면 화면이 세로에서 가로로 바뀐다.    http://cafefiles.naver.net/20101003_18/knossos88_1286111244696Ds2dV_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png        지금 당장 실행해보면 이런 그림이 나오지 않을 것이다. 그것은 당연히 대체 리소스를 작성하지 않았기 때문이다.    http://cafefiles.naver.net/20101003_195/knossos88_1286111313556yEK6N_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png        가로 방향일 경우를 지정하기 위해, 대체 리소스를 하나 더 만든다.      res 폴더 안에 layout\_land라는 이름으로 서브 폴더를 만들고, 이 안에 landport.xml을 하나 더 작성한다.      **Widget/layout-land/landport.xml**      <?xml version="1.0" encoding="utf-8"?>   <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"     **android:orientation="horizontal"**     android:layout\_width="fill\_parent"     android:layout\_height="fill\_parent"     android:gravity="center"     android:background="#e0e0e0"     >   <Button     android:layout\_width="100px"      android:layout\_height="wrap\_content"      android:text="누르세요"     />   <Button     android:layout\_width="100px"      android:layout\_height="wrap\_content"      android:text="수평이다"     />   </LinearLayout>    이제 실행해보면 결과가 위의 그림처럼 제대로 나올 것이다.    마지막으로 각 레이아웃은 위젯의 배치나 속성 값이 각각 다를 수 있지만, 위젯의 ID나 위젯 구성은 최소한 같아야 한다.      만약 리소스에 따라 위젯의 구성이 달라지면, 코드도 같이 달라져야 하므로 대체 리소스를 사용하는 의미가 없어진다.        **<텍스트 뷰>**    **기본 속성**      TextView는 안드로이드 위젯 중에 가장 기본이며, 이 클래스만 잘 연구해도 위젯의 공통적인 기능과 특성들을 대부분 파악할 수 있다.      문자열을 입출력하는 아주 간단한 기능을 수행하지만, 밖으로 보이는 기능보다 숨겨진 고급 기능들이 굉장히 많다.      TextView에는 문자열, 포맷팅, 출력, 입력, 편집에 관련된 대부분의 기능이 모두 구현되어 있다.      TextView 자체는 이들 기능의 대부분을 드러내지 않고 숨긴다.      잠시 후에 연구해 보겠지만, 버트과 에디트는 텍스트 뷰의 숨겨진 기능을 활성화하고 약간의 스타일을 입혔을 뿐이다.    http://cafefiles.naver.net/20101003_53/knossos88_1286113043274YBVF1_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      이제 하나하나 알아가보도록 하자.      텍스트 뷰가제공하는 속성 중 일부는 에디트에서 더 잘 확인할 수 있으므로 에디트도 같이 연구해 보자.      레이아웃 소스는 다음과 같다.      **Widget/textviewattr.xml**      그런데 소스가 너무 길어서, 파일로 첨부했다. 그저 res폴더 밑에 layout에 집어넣으면 된다.      (코드에서 불러오가민 하면 된다)      위젯의 갯수가 굉장히 많은데, 스크롤 뷰 안에 각각 조금씩 다른 텍스트 뷰와 에디트 뷰를 배치해 두었다.    http://cafefiles.naver.net/20101003_84/knossos88_1286113835156gHD6X_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      http://cafefiles.naver.net/20101003_141/knossos88_1286113975262NYOzT_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      처음과 끝만 캡처한 것인데, 소스를 실행시켜보면 알겠지만 이 가운데에 굉장히 많은 것들이 있다.      직접 실행시키고 눌러보며 어떤 차이가 있는지 느껴보기 바란다. 물론 밑에서 설명도 할 것이다.      **수평 입력**    위젯의 폭보다 더 긴 문자열이 입력된 경우의 동작을 지정한다.    http://cafefiles.naver.net/20101003_289/knossos88_1286114289102Lsnql_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      Normal : 자동개행. 폭을 넘어가면 알아서 개행해준다.    Scroll Horizontal : 사용자가 개행을 하지 않는 한, 계속해서 한 줄에 수평하게 입력한다. 그러나 분명 개행은 가능하다.    Single Line : 개행이 불가능하다. 입력을 얼마나 하든 끝까지 한 줄로 끝나게 된다. 즉, 개행불가.      **입력문자 제한**    http://cafefiles.naver.net/20101003_217/knossos88_12861147603583iMWE_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      digits : 지정한 문자열 외에 다른 문자를 집어넣을 수 없다. 위에는 aeiou 외에는 넣을 수 없을 것이다.    numeric : 숫자만 입력받는다. 다만 3가지 종류가 있다.      (1) integer : 아라비아 숫자만 입력가능    (2) signed : 선두에 -부호를 허용, 숫자 중간에는 올 수 없음    (3) decimal : 소수점을 허용, 소수점은 반드시 하나만 가능      만약 소수점과 음수 부호를 둘 다 허용하려면, decimal|signed라고 하면 된다.      phoneNumber : 전화번호를 구성하는 숫자, -    password : 입력 중인 문자를 감추고, 마지막 문자만 보여준다.      **커서 및 포커스**      간단하다.      selectAllOnFocus : 갖다대면 모두 선택된다.    Cursor Test - true : 커서가 보인다.    Cursor Test - false : 커서가 안보인다. 물론 문자입력이 가능하다.    http://cafefiles.naver.net/20101004_156/knossos88_1286150712308V7O77_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      **자동 링크**      autoLink 속성은 말 그대로다. 예를 들어 우리가 사이트 주소를 갖다붙이면 자동으로 링크가 생성되는 것처럼 말이다.      지원 가능한 링크는 다음과 같으며, | 연산자로 묶어 여러가지 링크를 같이 지정할 수도 있다.  http://cafefiles.naver.net/20101004_223/knossos88_1286151522551JJsIF_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png      **글자의 모양**      android:text="Shadow Text"   android:textSize="20px"   android:textColor="#ffffff"   android:shadowColor="#0000ff" // 그림자의 색상   android:shadowDx="3.0" // 글자와의 거리(x)   android:shadowDy="3.0" // 글자와의 거리(y)   android:shadowRadius="3.0" // 그림자의 크기    android:textScaleX="0.5" // 글자의 폭이다. 높이면 옆으로 늘어나고, 줄이면 날씬해진다.    http://cafefiles.naver.net/20101004_58/knossos88_1286151821605Ula1S_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png |

**<디버깅>**

  오류를 찾아 수정하는 것을 디버깅이라고 한다.

  디버깅을 하려면 먼저 Manifast에 다음 플래그를 지정해야 한다.

  <application android:icon="@drawable" android:label="@string/app\_name" android:debuggable="true">

  application 엘리먼트에 android:debuggable 속성을 true로 지정해야 디버깅이 가능하다.

  그러나 에뮬레이터는 원래 개발용 장비이므로, 이 속성이 없어도 디버깅이 가능하다.

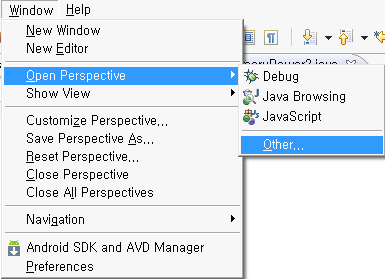
  하지만 실제 장비에는 이 속성을 지정해야 디버그용으로 컴파일 된다.

  또한 장비의 Setting 메뉴에서 USB Debugging 옵션을 켜야하며, 적절한 USB 드라이버가 설치되어 있어야 한다.

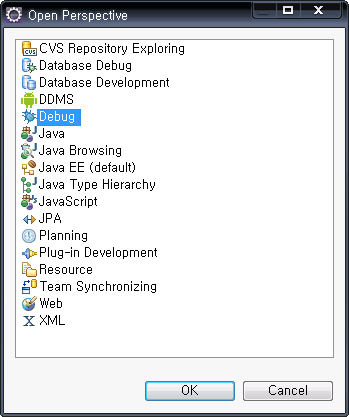
  아무래도 실장비는 최종 사용자용이다보니 디버깅을 위해 준비해야 할 것이 많다.

**디버그 실행하는 방법**

  (1) Window - Open Perspective - Other를 선택



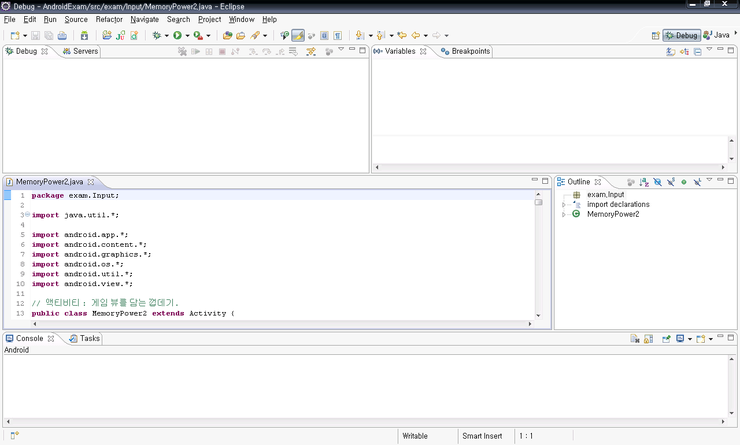
  (2) Debug를 클릭



  (3) 그리고 이제 우측 끝에 보면 Denug 모드가 생겼을 것이다. 누르면 된다.

http://cafefiles.naver.net/20100930_19/knossos88_1285836011050qc20U_png/%C1%A6%B8%F1_%BE%F8%C0%BD_knossos88.png

  (4) 그럼 대략 이런 화면이 나왔으리라



**단축키 설명**

  ■ F5 : 메서드 안으로 추적해 들어간다

  ■ F6 : 다음 명령문을 실행한다

  ■ F7 : 메서드의 끝가지 실행하고 리턴한다

  ■ F8 : 다음 중단점까지 실행한다

  ■ Ctrl + R : 커서가 있는 지점까지 실행한다

  ■ Ctrl + F2 : 디버깅을 중단한다

**<DDMS>**

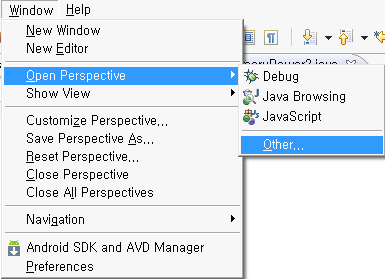
  DDMS(Dalvik Debug Monitor Service)는 ADT와 함께 설치되는 플러그인이며, 여러 가지 기능을 제공하는 디버깅 툴이다.

  포트포워딩, 화면 캡처, 스레드 및 힙 상태보기, 장비의 파일관리, 디버그 로그 보기, 통화 및 문자 메세지 테스트 등

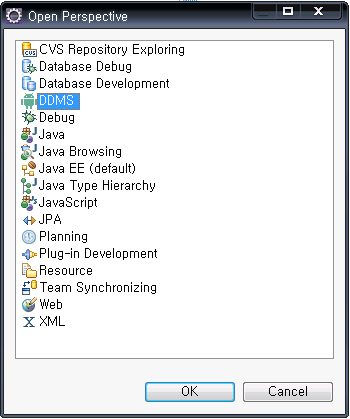
  에뮬레이터와 실제 장비에, 이런 테스트들을 할 수 있다.

**DDMS 실행하기**

  (1) Window - Open Perspective - Other를 선택



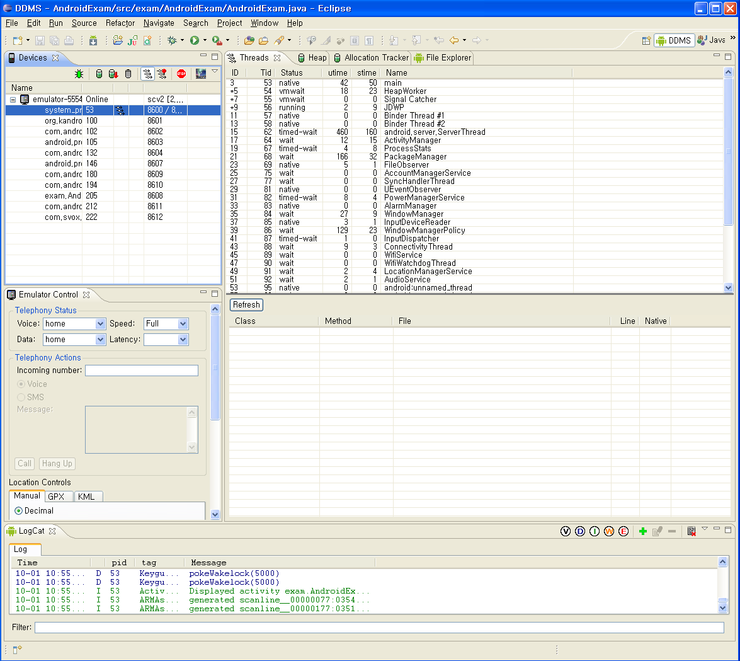
  (2) DDMS를 누른다



  (3) 그러면 이런 화면이 뜬다. DDMS 역시 Debug와 마찬가지로 우측상단에 버튼이 뜬다

       화면 왼쪽에는 연결된 장비의 목록과 실행 중인 프로세스 목록이 나타난다.

       안드로이드에서는 VM이 하나씩 할당되므로, 프로세스의 목록이 곧 VM의 목록과 같다.



  (4) 관리하고자 하는 프로세스를 목록에서 선택하고, 툴바에서 명령을 선택한다.



  실행 중인 프로세스에 디버거를 부착하여 디비깅을 할 수 있으며, 힙이나 스레드의 상태를 볼 수도 있다.

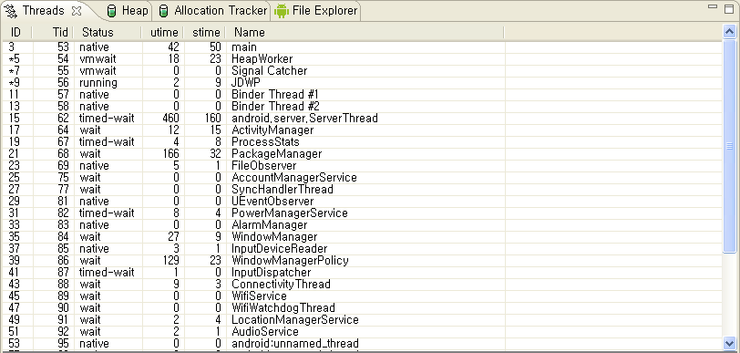
  프로세스 종료 명령은 프로그램을 강제로 종료한다.

  툴바 제일 오른쪽에는 화면 캡처 버튼이 있는데 누르면, 에뮬레이터나 장비의 화면을 이미지로 얻을 수 있다.

  오른쪽에는 여러 가지 정보가 탭별로 표시된다. Threads 탭에는 VM에서 실행중인 스레드의 목록이 표시된다.

  스레드 정보는 대단히 양이 많기 때문에, 매번 업데이트하면 네트워크 전송량이 너무 많아진다.

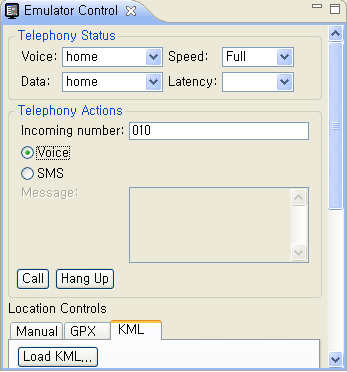
  그래서 UpdateThread를 눌러야만 스레드의 현재 상태, 사용 시간, 이름 등이 나타난다.



  Heap 탭에는 VM의 힙 정보가 표시되는데, 스레드와 마찬가지로 시간이 오래 걸리므로, 툴바의 Update Heap 버튼을 눌러야 갱신된다.

  아래 쪽의 Emulator Control 탭은 보다시피 전화나 SMS 문자 등을 에뮬레이션 한다.

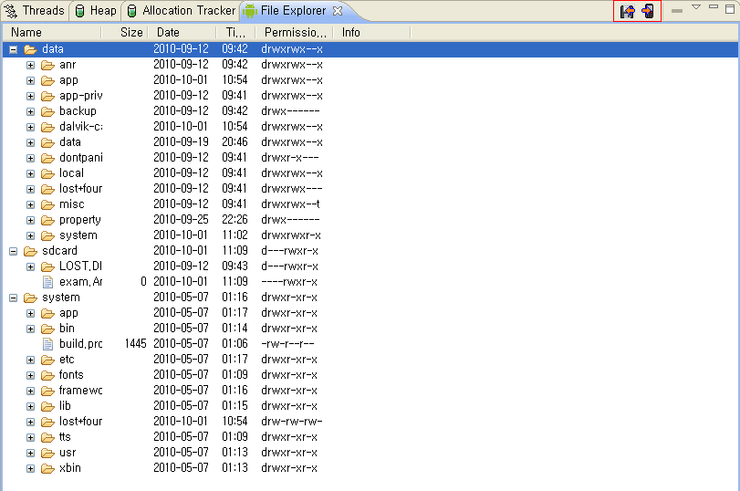
  한마디로 전화가 오거나 문자가 오게 할 수 있다.



  File Explorer 탭은 장비의 파일을 관리하는 툴이며, DDMS의 가장 실용적인 기능이다.

  안드로이드 장비 자체에 파일관리 탐색기가 내장되있지 않으므로, DDMS의 파일 탐색기를 통해서만 파일관리가 가능하다.

  중앙에 디렉토리 목록이 나타나고, 상단에 세 개의 파일 관리 버튼이 있다.



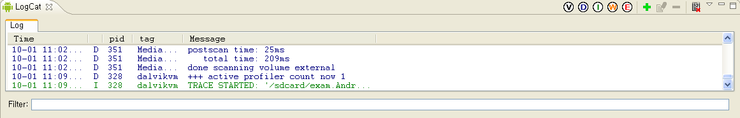
  우측 상단의 첫 번째 버튼은 장비의 파일을 컴퓨터로 복사하고, 두 번째 버튼은 컴퓨터의 파일을 장비로 복사한다.

  그 옆의 일자로 된 버튼은 삭제 버튼인데, 복구 기능이 없으므로 신중을 기해야 할 것이다.

  우리는 파일만 관리할 뿐, 디렉토리는 관리할 수 없다는 것을 알아두자.

  DDMS 하단에는 시스템 로그 정보가 표시된다. 로그 창에는 특정 프로세스가 시작되었다던가, 응용 프로그램을 설치한다거나,

  가비지 컬렉터가 쓰레기 수집 중이라는 둥 장비에서 일어나는 사건들이 상세하게 보고된다.



  응용프로그램의 필요에 따라 로그를 남길 수 있는데, 그것은 다음 항에서 보도록 하자.

**<LogCat>**

  로그는 시스템이나 응용 프로그램의 동작에 대한 상세한 기록이다.

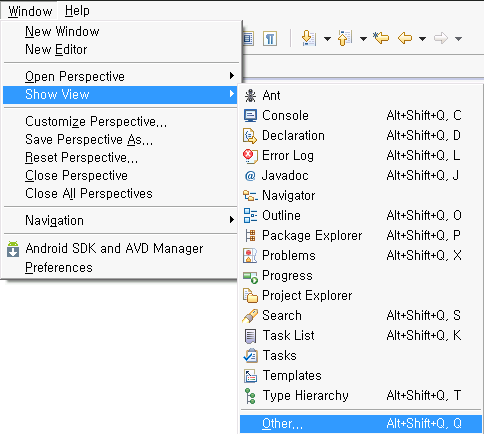
  디버깅은 느린데다 직접 해봐야 상태를 알 수 있는데, 로그는 항상 출력되며 또 실시간으로 출력된다는 장점이 있다.

  멀티 스레드나 네트워크 응용 프로그램 같은 복잡한 문제도 로그를 사용하면 쉽게 발견할 수 있다.

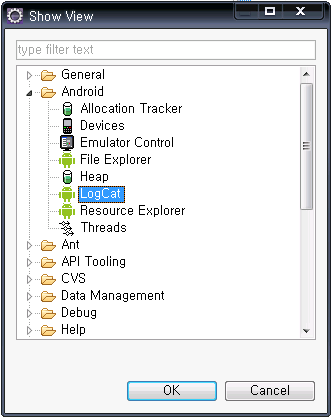
  따라서 가급적이면 로그 출력문을 많이 작성해 놓는 것이 좋다.

**LogCat을 실행하는 방법**

  (1) Window - Show View - Other를 선택



  (2) Android에서 LogCat을 선택한다.



  (3) 이렇게 LogCat이 실행된 것을 확인할 수 있다.



**로그 출력 메서드** : 로그를 출력할 때는 다음 메서드를 활용한다

  디버깅용, 실행시 유효하지 않음

  static int Log.v(String tag, String msg, Throwable tr) : 개발 중에만 컴파일 된다 (Verbose)

  static int Log.d(String tag, String msg, Throwable tr) : 실행 중에는 제거된다 (Debug)

  항상 유지되는 로그

  static int Log.e(String tag, String msg, Throwable tr) : 심각한 상황 (Error)

  static int Log.w(String tag, String msg, Throwable tr) : 경미한 경고 (Warning)

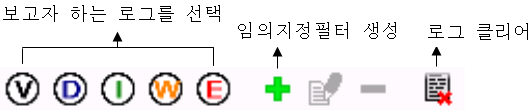
  static int Log.i(String tag, String msg, Throwable tr) : 단순한 정보 (Information)

  어렵게 생각할 필요가 없이 (Throwable도 필요없다)

**Log..v(TAG,"Test Message");** // 당연히 v말고도 원하는 로그로 바꾸어 쓰면 된다.

  이렇게만 적으면 된다. static이므로 정적메서드, 그 말인즉 별도의 객체가 필요없이 바로 쓰면 된다는 얘기다.

**버튼 설명**



  말 그대로 5개의 버튼은 원하는 로그만을 선택해 볼 수 있고,

  로그 클리어는 로그 메세지가 너무 많아 혼동스러울 경우, 클리어를 통해 기존 메세지를 모두 날려버릴 수 있다.

  임의지정 필터는 내가 단독으로 보고 싶을 경우에 생성한다.

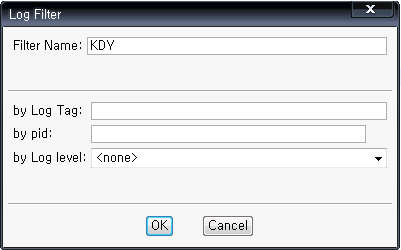
  예를 들어, 아까 **Log..v(TAG,"Test Message");** 의 경우에 너무 많다 싶으면

**Log..v("KDY","Test Message");** 이렇게 단독으로 뽑아 볼 수 있다는 것이다.

  그러려면 먼저 단독으로 사용할 로그 필터를 생성해야 한다.

  (1) 초록색 '필터추가'를 누른다.

  (2) 그러면 이런 화면이 나오고, 여기에 이름만 적으면 된다.



  (3) 이렇게 단독으로 필터가 추가되었다. 사용자가 정의한 필터를 이쪽으로 오게 되있다.



**<adb>**

  adb(Android Debug Bridge)는 명령 행에서 장비의 상태를 관리하는 툴이다.

  그러나 adb의 기능 대부분이 DDMS에서 사용이 가능하므로 자주 사용되지 않는다.

  (다만, 우리는 한글 키보드 설치편에서 adb를 사용한 적이 있다)

  그러나 DDMS가 지원하지 못하는 고급 기능과 스크립트로 자동화 할 수 있다는 것은 장점이다. (일단 모르더라도 끄떡거려라)

  adb는 다음 세 요소가 협력적으로 동작하여 개발 컴퓨터와 장비 간의 통신을 수행한다.

  ■ 클라이언트 : 개발 컴퓨터에서 실행되는 관리 툴, adb가 대표적.

  ■ 서버 : 개발 컴퓨터에서 백그라운드로 실행, 클라이언트와 데몬을 중계하는 역할. TCP 5037 포트를 통해 클라이언트와 통신

  ■ 데몬 : 장비에서 백그라운드로 실행, 클라이언트로 전달된 명령을 장비에서 수행하는 역할.

**<메뉴>**

  메뉴는 그래픽 위주의 모바일 환경과는 잘 어울리지 않지만, 아주 오래 전 부터 사용자의 명령을 받아들이는

  가장 일반적인 인터페이스로 활용되어 왔다.

  안드로이드의 메뉴는 그다지 세련되지 못해서 최종 제품에 채용하기는 망설여지지만

  테스트용이나 설정 조정 등의 용도로는 무난하게 쓸만하다.

  화면에 직접 보이지 않으므로, 특수한 기능을 살짝 숨겨 놓을 수도 있다.

  안드로이드가 지원하는 메뉴는 다음 세 가지 종류가 있다.

   · 옵션 메뉴 : 액티비티의 주 메뉴, 평소에는 화면 아래쪽에 숨겨져 있으나 MENU 버튼을 누르면 활성화가 된다.

                     최대 6개까지 표시가 가능하고, 6개가 넘으면 나머지는 More 항목  안에 숨어있다가 확장 메뉴 형태로 나타난다.

                     아이콘을 달아 장식할 순 있지만 체크 박스나 라디오를 구성할 수는 없고, 단순히 명령을 입력받을 수만 있다.

   · 컨텍스트 메뉴 : 화면을 일정 시간동안 누르고 있으면(Hold Down) 화면 중앙에 나타난다.

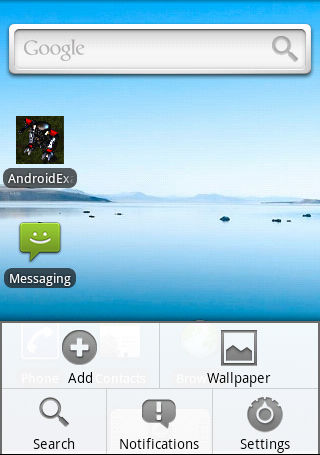
                           주로 리스트 뷰에 사용되나, 임의의 위젯에 대해서도 사용할 수 있다.

                           체크 박스나 라디오 그룹으로 만들 수 있다.

   · 서브 메뉴 : 옵션 메뉴가 꽉 차서, 더 많은 메뉴 항목을 넣을 싶을 때 사용한다.

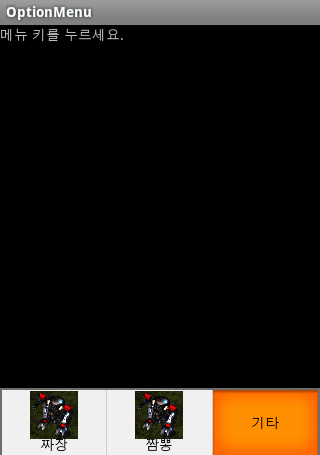
                     서브 메뉴 안에 또 다른 서브 메뉴를 중첩할 수 없으므로 무한히 생성할 수는 없다.

  다시 한마디로 정리하도록 하겠다.



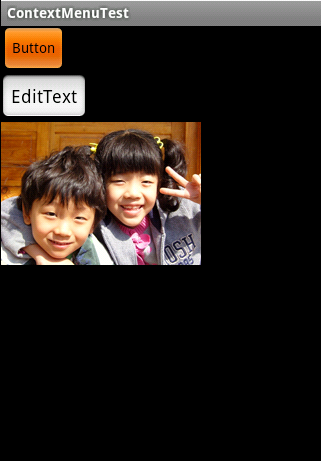
  키패드의 MENU 버튼을 눌렀을 때, 이렇게 밑에서 나오는 메뉴를 옵션 메뉴라 한다.

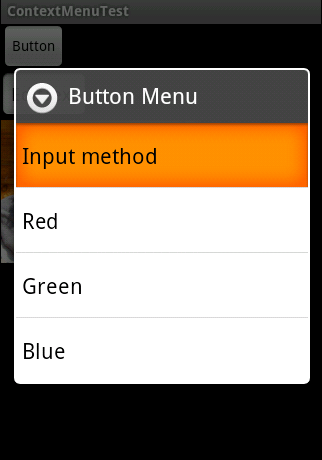
  그리고 나머지 서브 메뉴와 컨텍스트 메뉴는 헷갈려하시는 분들이 있는데, 그 이유는 모양이 똑같기 때문이다.

  이 차이는 간단하다. 옵션 메뉴로 들어간 메뉴는 서브 메뉴, 롱 클릭으로 들어간 메뉴를 컨텍스트 메뉴라 한다. 



  이렇게 옵션 메뉴를 통해 들어가게 되면, 서브 메뉴라고 하고

  컨텍스트 메뉴는 



  버튼을 long-click 하거나, 키패드의 중앙 버튼을 long-click 하게 되면, 똑같이 메뉴가 나타난다.

  이것을 우리는 컨텍스트 메뉴라고 한다.

  다만 컨텍스트는 단순히 버튼 뿐만이 아니라, 포커스 입력 부분, 이미지까지 다양한 뷰에 대해 설정할 수가 있다.

  이는 예제를 통해 더 알아보도록 하자.

**<옵션 메뉴>**

  가장 일반적이고 자주 사용되는 옵션 메뉴부터 만들어 보자.

  장비의 MENU 버튼을 누르면 액티비티의 다음 콜백 메서드가 호출되는데, 이 매서드를 재정의하여 옵션 메뉴를 생성한다.

**boolean onCreateOptionsMenu(Menu menu)**

  인수로 빈 메뉴 객체인 menu가 전달되는데, 여기에 메뉴 항목을 추가한다.

  메뉴 항목을 추가할 때는 add 메서드를 호출한다.

**MenuItem add(int groupid, int itemid, int order, CharSequence title)**

**MenuItem add(int groupid, int itemid, int order, int order, int titleRes)**

**SubMenu addSubMenu(int titleRes)**

  어렵게 생각할 필요가 없다.

  Menuitem(그룹ID, 개인ID, 우선순위, 메뉴이름)

  SubMenu addSubMenu(서브메뉴이름)

  이론적으로 보면 어렵지만, 잠시 후에 예제와 함께 같이 본다면 어렵지 않을 것이다.

  MenuItem 클래스의 다음 메서드들로 항목의 상세 속성들을 지정한다.

**MenuItem setIcon(int iconRes) : 말 그대로 메뉴 항목과 같이 출력되는 작은 이미지**

**MenuItem setTitle(Char sequence title) : 역시 말 그대로 제목을 정하는 것**

**MenuItem setAlphabeticShortcut(char alphaChar) : (알파벳) 단축키를 설정한다**

**MenuItem setNumericShortcut(char numericChar) : (숫자) 단축키를 설정한다**

**MenuItem setChecked(boolean checked) : 미리 체크된 상태를 적용할 것인지 여부를 묻는 것이다**

**MenuItem setEnabled(boolean enabled) : 활성화 여부를 결정한다. 버튼의 경우엔 클릭을 할 수 없다.**

**MenuItem setVisible(boolean visible) : 보일 것인지, 말 것인지를 결정한다. 사용하면 메뉴 선택은 할 수 없다.**

  이 메서드들은 별도의 변수에 대입할 필요없이, 연속적으로 조정이 가능하다.

  일례로 menu.add(...).setIcon(...).setTitle(...).setChecked(...);

  이렇게 '.'을 통해 연속적인 설정이 가능하다.

  앞서 onCreateMenu 메서드는 최초 메뉴가 만들어질 때 한번만 호출되며 다시는 호출되지 않는다.

  따라서 매뉴는 한번만 초기화되며 이 메서드에서 만든 메뮤가 계속 사용된다.

  만약 실행 중에 메뉴를 편집하려면 다음 메서드를 재정의한다.

**boolean onPrepareOptionsMenu(Menu menu)**

  이 메서드는 메뉴가 열릴 때마다 호출되므로, 메뉴 항목을 더 추가하거나 뺄 수 있고 메뉴 항목의 속성을 변경할 수도 있다.

  메뉴 항목이 선택될 때는 다음 메서드가 호출된다.

**boolean onOptionsItemSelected (MenuItem item)**

  인수로 사용자가 선택한 메뉴  항목 개체가 들어가게 된다.

  item의 getItemId 메서드를 호출하면 항목의 ID를 조사할 수 있으며, ID에 따라 적당한 명령을 수행하면 된다.

**Input/OptionMenu.java**

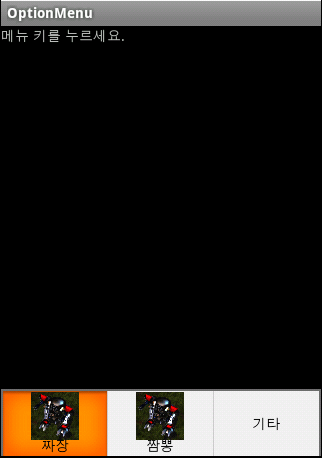
  public class OptionMenu extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      TextView MyText = new TextView(this);  
      MyText.setText("메뉴 키를 누르세요.");  
      setContentView(MyText);  
    }

    public boolean onCreateOptionsMenu(Menu menu) {  
      super.onCreateOptionsMenu(menu);  
      MenuItem item=menu.add(0,1,0,"짜장");  
      item.setIcon(R.drawable.icon);  
      item.setAlphabeticShortcut('a');  
      //menu.add(0,1,0,"짜장").setIcon(R.drawable.icon).setAlphabeticShortcut('a');  
      menu.add(0,2,0,"짬뽕").setIcon(R.drawable.icon);  
      SubMenu etc = menu.addSubMenu("기타");  
      etc.add(0,3,0,"우동");  
      etc.add(0,4,0,"만두");

      return true;  
    }

    public boolean onOptionsItemSelected(MenuItem item) {  
      switch (item.getItemId()) {  
        case 1:  
          Toast.makeText(this,"짜장은 달콤해",Toast.LENGTH\_SHORT).show();  
          return true;  
        case 2:  
          Toast.makeText(this,"짬뽕은 매워",Toast.LENGTH\_SHORT).show();  
          return true;  
        case 3:  
          Toast.makeText(this,"우동은 시원해",Toast.LENGTH\_SHORT).show();  
          return true;  
        case 4:  
          Toast.makeText(this,"만두는 공짜야",Toast.LENGTH\_SHORT).show();  
          return true;  
      }  
      return false;  
    }  
  }

  아래 그림처럼 키패드의 MENU를 누른 후 짜장을 선택하면, '짜장은 달콤해"라는 토스트를 띄워줄 것이다.

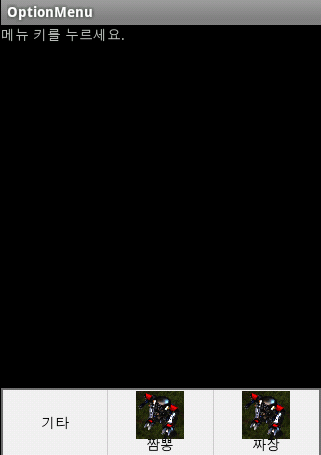


  위에서 보면 menu.add(0,1.0,"짜장"); 이라고 선언했다.

**MenuItem add(int groupid, int itemid, int order, CharSequence title)**

전혀 어렵지 않다. 일단은 그룹짓지 않았기에 0, 자신의 아이디로 1을 주었고, 우선순의는 주지 않았다. 제목은 짜장이다.

  menu.add와 MenuItem add를 비교해보면 확연히 감이 올 것이다.



  이것은 우선순위를 준 것이다.

  menu.add(0,1.2,"짜장"); // 짜장의 우선순위를 2순위로 두고

  menu.add(0,2,1,"짬뽕"); // 짬뽕의 우선순위를 1순위로 두었더니 자리가 바뀌었다.

  기타 메뉴는 디폴트라 0 순위로 자연히 올라왔다.

  SubMenu etc = menu.addSubMenu("기타"); // 이렇게 설정했지만

  서브 메뉴도 SubMenu etc = menu.addSubMenu(0,3,3,"기타"); // 이렇게 우선순위를 줄 수도 있다.

  또한 이 항목은 menu.add(0,1,0,"짜장").setIcon(R.drawable.icon).setAlphabeticShortcut('a');

  아까 우리가 앞서 말했던 연속적 속성 설정이 가능하다는 것을 보여주고 있다. 모두 '.'으로 속성을 이어가고 있다.

  이번 OptionMenu 예제에서는

**boolean onOptionsItemSelected (MenuItem item)**만 있고, **boolean onPrepareOptionsMenu(Menu menu)**는 없는데

  그 이유는 말 그대로 Prepare는 딱히 필요가 없다. 우리가 원하는 것은 선택했을 때의 설정이 바뀌는 것을 원하기 때문이다.

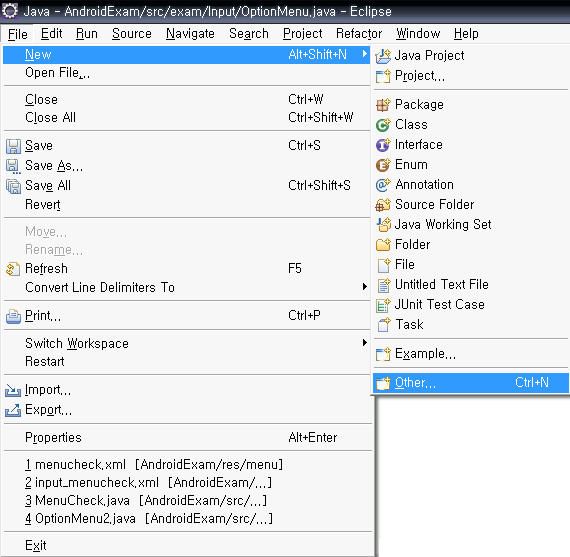
**<XML로 메뉴 정의하기>**

  메뉴도 일종의 디자인이기 떄문에, 코드보다 XML로 작성하는 것이 훨씬 더 쉽고 관리하기도 좋다.

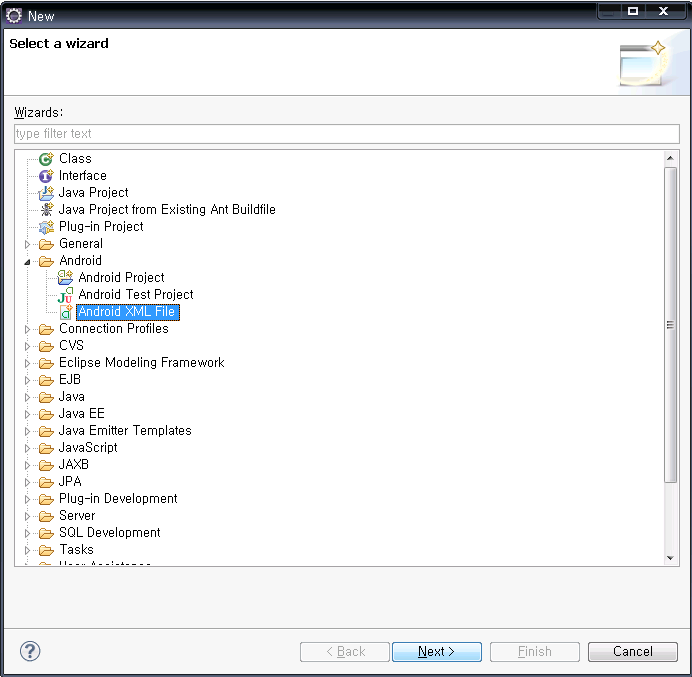
  메뉴의 모양을 정의하는 XML 파일은 시스템이 만들어주지 않으므로, res 폴더 밑에 menu라고 별도로 생성해야 한다.

  이것은 별도로 폴더를 생성하지 않고, 프로젝트를 생성하는 방법이다.

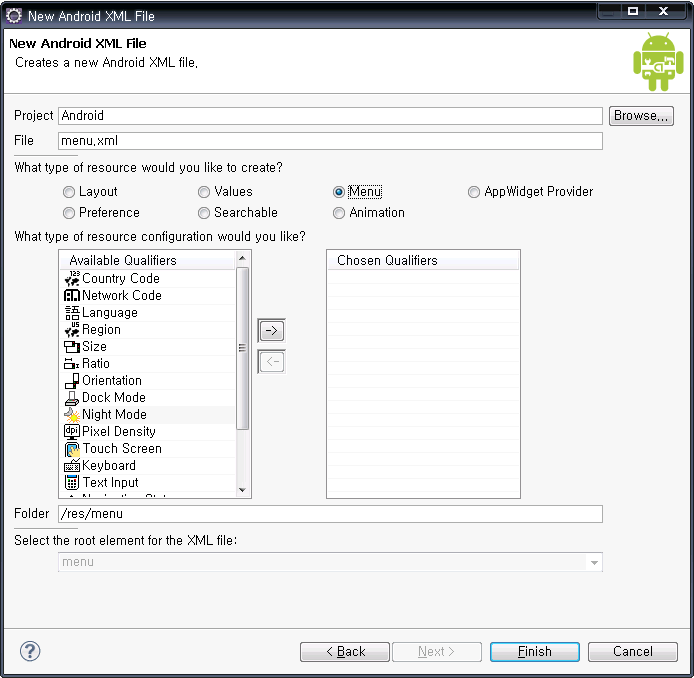
  (1) File - New - Other를 클릭



  (2) Android XML File을 선택하고, Next를 누른다.



  (3) File 이름을 menu.xml로 설정하고, Type을 Menu로 설정하면 알아서 폴더가 생성된다. Finish를 누르자.



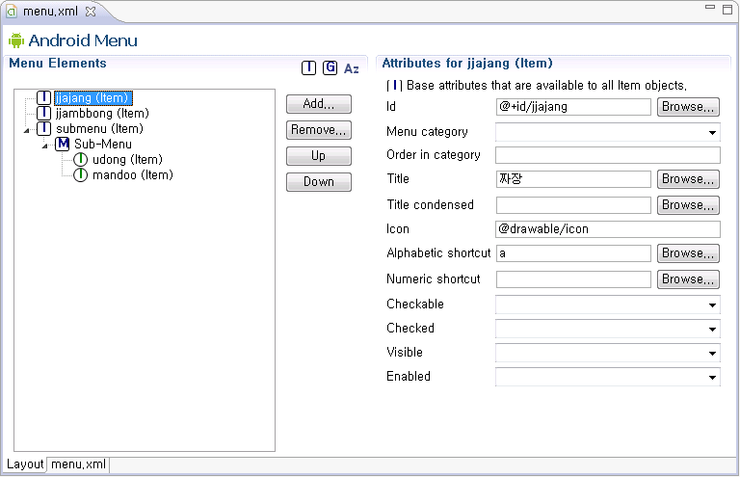
  자, 이제 만들어진 XML 파일에 다음과 같이 작성해보자.

**Input/menu.xml**

  <menu xmlns:android="<http://schemas.android.com/apk/res/android>">  
    <item android:id="@+id/jjajang"  
      android:title="짜장"  
      android:icon="@drawable/icon"  
      android:alphabeticShortcut="a"  
      />  
    <item android:id="@+id/jjambbong"  
      android:title="짬뽕"  
      android:icon="@drawable/icon"   
      />  
    <item android:id="@+id/submenu"  
      android:title="기타">  
      <menu>  
        <item android:id="@+id/udong"  
          android:title="우동"  
          />  
        <item android:id="@+id/mandoo"  
          android:title="만두"  
          />  
      </menu>  
    </item>  
  </menu>

  각 속성의 이름은 따로 설명하지 않더라도 어떤 의미인지 쉽게 알 수 있을 것이다.

  문서 창의 아래쪽에 Layout 탭을 선택하면 다음과 같은 메뉴 편집기가 열린다.



  왼쪽에는 메뉴의 구조가 계층적으로 표시되며, 오른쪽에서 각 메뉴 항목의 속성을 편집한다.

  XML 메뉴를 생성할 때는 MenuInflate 객체의 inflate 메서드를 사용하여 전개한다.

  메뉴 전개자인 MenuInflater 객체는 Activity.getMenuInflater 메서드로 구한다. 인수로 메뉴의 리소스 ID만 전달하면 된다.

**Input/OptionMenu2.java**

  public class OptionMenu2 extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      TextView MyText = new TextView(this);  
      MyText.setText("메뉴 키를 누르세요.");  
      setContentView(MyText);  
    }

    public boolean onCreateOptionsMenu(Menu menu) {  
      **MenuInflater inflater = getMenuInflater();** // MenuInflater 객체를 얻어온 다음에  
      **inflater.inflate(R.menu.menu,menu);** // 그것을 전개시킨다  
      return true;  
    }  
      
    public boolean onOptionsItemSelected(MenuItem item) {  
      switch (item.getItemId()) {  
        case R.id.jjajang:  
          Toast.makeText(this,"짜장은 달콤해",Toast.LENGTH\_SHORT).show();  
          return true;  
        case R.id.jjambbong:  
          Toast.makeText(this,"짬뽕은 매워",Toast.LENGTH\_SHORT).show();  
          return true;  
        case R.id.udong:  
          Toast.makeText(this,"우동은 시원해",Toast.LENGTH\_SHORT).show();  
          return true;  
        case R.id.mandoo:  
          Toast.makeText(this,"만두는 공짜야",Toast.LENGTH\_SHORT).show();  
          return true;  
       }  
     return false;  
    }  
  }

  메뉴 항목을 생성하고 속성을 설정하는 코드가 모두 제거되었으므로 onCreateOptionsMenu 메서드의 길이가 대폭 짧아졌다.

  예제의 동작은 코드로 만들었을 때와 완전히 동일히다. 어차피 XML 파일은 메뉴의 구조만 기술할 뿐, 동작을 정의하진 않기 때문이다.

  개발 과정도 편리할 뿐만 아니라 국제화에도 유리하므로, 이후 메뉴를 디자인 할 때는 가급적 XML을 사용하길 권장한다.

  왜냐면 한국어, 영어, 일본어 등 각 언어별로 메뉴를 작성해 두면 시스템이 장비의 언어에 맞는 메뉴를 자동으로 Load하므로 편리하다.

**<체크 메뉴 항목>**

  말 그대로, 체크 메뉴 항목을 말한다.

  컨텍스트 메뉴와 서브 메뉴에만 사용이 가능하다. 옵션 메뉴는 사용할 수 없는 대신 아이콘을 달 수 있다.

  레이아웃 버튼 하나를 배치해 놓고 폰트의 크기와 색상을 옵션으로 입력받아 실행 중에 변경해 볼 것이다.

**Input/menucheck.xml**

  <menu xmlns:android="<http://schemas.android.com/apk/res/android>">  
    <item android:id="@+id/submenu"  
      android:title="Sub Menu">  
      <menu>  
        <group android:id="@+id/checkable\_group"  
          android:checkableBehavior="**all**">  
          <item android:id="@+id/bigfont"  
            android:title="BigFont" />  
        </group>  
        <group android:id="@+id/exclusive\_checkable\_group"  
          android:checkableBehavior="**single**">  
          <item android:id="@+id/red"  
            android:title="Red"   
            android:checked="true" />  
          <item android:id="@+id/green"  
            android:title="Green" />  
          <item android:id="@+id/blue"  
            android:title="Blue" />  
        </group>  
      </menu>  
    </item>  
  </menu>

  서브 메뉴 안에 2개의 그룹을 정의했는데, checkableBehavior 속성에 따라 체크 상태를 관리하는 방법과 모양이 달라진다.

  이 속성의 의미는 다음과 같은데, 도움말에는 설명이 완전히 반대로 작성되어 있으며 2.1까지도 수정이 안되고 있다.

**checkableBehavior 속성**

  ■ none : 체크를 할 수 없는 단독 명령이다

  ■ all : 모든 항목을 개별적으로 선택할 수 있으며, 항목 옆에 네모난 체크 박스가 표시된다

  ■ single : 그룹에 속한 항목 중 하나만 배탄적으로 선택할 수 있으며, 항목 옆에 동그란 버튼이 표시된다

  코드는 다음과 같으며, 메뉴와 관련도니 세 개의 메서드를 모두 재정의했다.

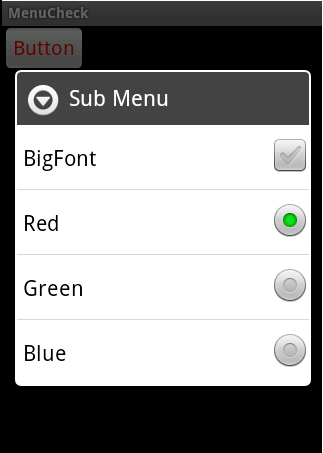
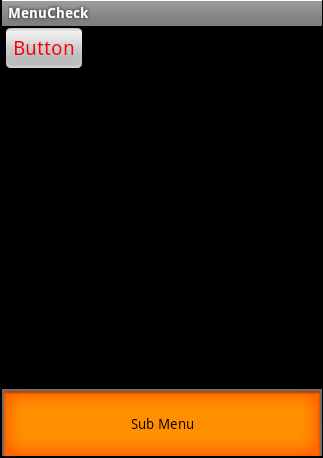
**menu/menucheck.xml**

  public class MenuCheck extends Activity {  
    Button mBtn;  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.input\_menucheck);  
      mBtn = (Button)findViewById(R.id.button);  
    }

    public boolean **onCreateOptionsMenu**(Menu menu) {  
      MenuInflater inflater = getMenuInflater();  
      inflater.inflate(R.menu.menucheck,menu);   
      return true;  
    }  
    public boolean **onPrepareOptionsMenu**(Menu menu) {  
      if (mBtn.getTextSize() == 40) {  
        menu.findItem(R.id.bigfont).setChecked(true);  
      } else {  
        menu.findItem(R.id.bigfont).setChecked(false);  
      }   
      int color = mBtn.getTextColors().getDefaultColor();  
      if (color == Color.RED) {  
        menu.findItem(R.id.red).setChecked(true);  
      }  
      if (color == Color.GREEN) {  
        menu.findItem(R.id.green).setChecked(true);  
      }  
      if (color == Color.BLUE) {  
        menu.findItem(R.id.blue).setChecked(true);  
      }  
      return true;  
    }   
    public boolean **onOptionsItemSelected**(MenuItem item) {  
      switch (item.getItemId()) {  
        case R.id.bigfont:  
          if (item.isChecked()) {  
            mBtn.setTextSize(20);  
          } else {  
            mBtn.setTextSize(40);  
          }  
          return true;  
        case R.id.red:  
          mBtn.setTextColor(Color.RED);  
          return true;  
        case R.id.green:  
          mBtn.setTextColor(Color.GREEN);  
          return true;  
        case R.id.blue:  
          mBtn.setTextColor(Color.BLUE);  
          return true;  
      }  
      return false;  
    }  
  }

  onCreateOptionsMenu 메서드는 최초 메뉴를 생성할 때, 딱 한번만 호출되는데 비해

  onPrepareOptionsMenu 메서드는 메뉴를 열 때마다 호출되므로, 이 메서드에서 현재 상태에 따른 속성을 조정한다.



  메뉴 리소스에 초기화된 대로 BigFont 옵션은 선택되어 있지 않고, 빨간색으로 초기화가 되어 있다.

  메뉴를 통해 선택을 변경하면, 버튼 텍스트의 크기와 색상이 바뀌며 변경한 옵션은 메뉴의 체크 박스·라디오 버튼에도 표시된다.

**<컨텍스트 메뉴>**

  컨텍스트 메뉴는 특정 뷰나 항목에 필요한 명령들만 모아 놓은 메뉴이다.

  한마디로, long-click으로 열거나 센터 키패드로 여는 모든 것은 컨텍스트 메뉴다.

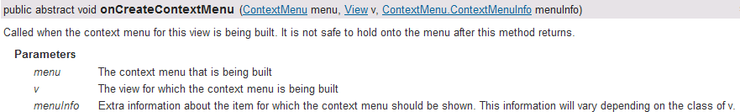
  컨텍스트 메뉴를 달고 싶은 뷰는 다음 메서드로 등록해야 한다.

**void Activity.registerForContextMenu (View view)**

  등록해놓으면 View.OnCreateContextMenuListener가 액티비티 자신으로 등록되고, 뷰 위에서 다음 메서드를 호출한다.

**void OnCreateMenu (ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)**

  이게 뭔가하고 궁금해하시는 분들을 위해 API를 직접 가져왔다.



  가끔 이 형식이 뭔가, 메서드, 인수 하나하나 다 파고드시는 분들이 계신데 다 부질 없는 짓이다.

  자세하게 공부하는 것도 중요하지만, 그렇게 하다보면 자바나 안드로이드나 끝이 없다.

  onCreateContextMenu는 그냥 말 그대로 컨텍스트 메뉴를 만들어주는 메서드이다.

  여러 개의 뷰를 등록해 놓을 수 있으며, 메서드 v에는 꼭 필요한 항목을 등록하면 된다.

  menuinfo는 표시해야 할 메뉴에 대한 세부 정보인데, 뷰에 따라 제공되는 정보는 달라진다.

  옵션 메뉴와 달리, onCreateContextMenu 메서드는 메뉴가 필요할 때마다 매번 호출된다.

  메뉴는 항상 새로 생성되며, 상황에 따라 메뉴 항목의 개수나 속성을 원하는 대로 지정할 수 있다.

  따라서 메뉴의 속성을 변경하는 메서드는 따로 제공하지 않는다.

  Context 메뉴의 항목을 선택하면 다음 메서드가 호출된다.

**boolean onContextItemSelected (MenuItem item)**

  말 그대로, 선택되었을 때의 동작을 기술하면 된다. 잠시 뒤 우리는 이 메서드를 switch문제 적절하게 사용할 것이다.

  조금 위애서 onCreateContextMenu를 보았는데, 그것은 액티비티에서 뷰의 컨텍스트 메뉴를 통합 관리하는 방법이고

  이번에는 뷰 자체에서 컨텍스트 메뉴를 직접 생성하는 방법을 보겠다.

**void onCreateContextMenu (ContextMenu menu)**

  아까보다 인수가 줄었다. 당연한 것이 이것은 뷰 자체이므로 View v를 전달할 필요가 없다.

  ■ 장점 : 자신의 메뉴를 스스로 정의한다는 면에서 캡슐화에 유리

  ■ 단점 : 이 메서드를 재정의하려면 커스텀 위젯을 상속받아야 한다는 점에서 번거롭다

  다음 예제는 세 개의 위젯에 대해 각각 다른 컨텍스트 메뉴를 표시한다.

  레이아웃에는 3개의 위젯이 배치되어 있다.

**Input/ContextMenu.java**

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    >  
  <Button  
    android:id="@+id/button"  
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"  
    android:text="Button"  
    />  
  <EditText  
    android:id="@+id/edittext"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:text="EditText"  
    />  
  <exam.Input.MyImage    
    android:id="@+id/myimage"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:src="@drawable/childs"  
    />  
  </LinearLayout>

  MyImage는 소스에서 별도로 정의한 커스템 위젯이다.

  위젯과 마찬가지로 XML 문서에서 사용할 수 있되, 패키지명을 완전하게 적어야 한다는 점이 다르다.

  drawable 폴더에는 이미지 뷰에 출력할 childs라는 이미지를 넣어 두었다.

  소스는 다음과 같다.

**Input/ContextMenu.java**

  public class ContextMenuTest extends Activity {  
    Button mBtn;  
    EditText mEdit;  
    MyImage mImage;  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.input\_contextmenu);  
          
      mBtn = (Button)findViewById(R.id.button);

**registerForContextMenu(mBtn);**          
      mEdit = (EditText)findViewById(R.id.edittext);  
      **registerForContextMenu(mEdit);**

      mImage = (MyImage)findViewById(R.id.myimage);  
      **registerForContextMenu(mImage);**    }  
      
    public void **onCreateContextMenu** (ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {  
      super.onCreateContextMenu(menu, v, menuInfo);  
       
      if (v == mBtn) {  
        menu.setHeaderTitle("Button Menu");  
        menu.add(0,1,0,"Red");  
        menu.add(0,2,0,"Green");  
        menu.add(0,3,0,"Blue");  
      }

      if (v == mEdit) {  
        menu.add(0,4,0,"번역하기");  
        menu.add(0,5,0,"필기 인식");  
      }  
    }  
      
    public boolean onContextItemSelected (MenuItem item) {  
      switch (item.getItemId()) {  
        case 1:  
          mBtn.setTextColor(Color.RED);  
          return true;  
        case 2:  
          mBtn.setTextColor(Color.GREEN);  
          return true;  
        case 3:  
          mBtn.setTextColor(Color.BLUE);  
          return true;  
        case 4:  
          Toast.makeText(this,"번역했다.",Toast.LENGTH\_SHORT).show();  
          return true;  
        case 5:  
          Toast.makeText(this,"필기 인식했다.",Toast.LENGTH\_SHORT).show();  
          return true;  
        case 100:  
          Toast.makeText(this,"회전했다 치고.",Toast.LENGTH\_SHORT).show();  
          return true;  
        case 101:  
          Toast.makeText(this,"크기 변경 했다 치고.",Toast.LENGTH\_SHORT).show();  
          return true;  
      }   
      return true;  
    }  
  }

  class MyImage extends ImageView {  
    public MyImage(Context context) {  
      super(context);  
    }  
   
    public MyImage(Context context, AttributeSet attrs) {  
      super(context, attrs);  
    }  
   
    public void onCreateContextMenu(ContextMenu menu) {  
      super.onCreateContextMenu(menu);

      menu.setHeaderTitle("MyImage Menu");  
      menu.add(0,100,0,"이미지 회전");  
      menu.add(0,101,0,"크기 변경");  
    }  
  }

  위에서 보면 onCreate에서 3개의 위젯에 대해 컨텍스트 메뉴를 등록했다.

  사용자가 각 위젯을 long-click하게 되면, onCreateContextMenu 메서드가 호출되며, 두 번째 인수로 long-click한 변수가 전달된다.

  메서드 내에서는 컨텍스트 메뉴를 요청한 뷰가 누구인가에 따라 필요한 명령을 menu에 추가한다.

  커스텀 뷰는 onCreateMenu를 재정의하여 컨텍스트 메뉴를 직접 생성한다.

  그러나 컨텍스트 메뉴는 직접 생성하더라도, 선택시의 동작은 직접 할 수 없다.

  이는 onContextItemSelected 메서드에서 한꺼번에 수행해야 한다.

  버튼의 컨텍스트 메뉴를 통해 색상을 빠르고 편리하게 선택했으나, 이는 사용자들이 컨텍스트의 메뉴를 인식했을 경우다.

  외형만 봐서는 컨텍스트 메뉴가 있는지 알 수 없으므로, 메뉴얼이나 도움말을 통해 미리 알려주어야 한다.

**<종합 테스트>**

  지금까지 많이 배운 것은 아니지만, 그렇다고 결코 적게 배우지는 않았다.

  여기까지 학습한 내용은 지극기 기초적이며, 중요한 것들이어서 얼마나 이해했는가에 따라 차후 학습 곡선이 달라진다.

  이쯤에서 단편적인 기법들만 볼 것이 아니라, 완성된 예제를 만들어 보고 배운 것들을 정리해보자.

**기억력 게임**

  이 게임은 화면에 새로 나타나는 도형을 선택하는 것이다. 만약 마지막 도형을 찾지 못한다면 게임은 끝나게 되며

  다시 할지 말지를 묻는 메세지를 출력해준다. 다시 한다면 게임을 초기화하고, 아니라면 메뉴로 돌아간다.

  사용자 액션이 단순하고, 논리의 흐름도 직관적이라 분석용 예제로 적합하다. 레이아웃이나 리소스 없이 코드만 보면 된다.

**Input/MemoryPower**

  // 액티비티 : 게임 뷰를 담는 껍데기.  
  public class MemoryPower extends Activity {  
    GameView gv;

    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      gv = new GameView(this);  
      setContentView(gv);  
    }  
   
    //도형 하나에 대한 정보  
    class Shape {  
      final static int RECT = 0;  
      final static int CIRCLE = 1;  
      final static int TRIANGLE = 2;

      int what;  
      int color;  
      Rect rt;  
    }

    // 게임 뷰 : 실질적인 메인이다.  
    class GameView extends View {

      // 그리기 모드. 빈화면 또는 도형 출력  
      final static int BLANK = 0;  
      final static int PLAY = 1;

      // 게임 진행 속도  
      final static int DELAY = 1500;

      // 현재 그리기 모드  
      int status;

      // 생성된 도형의 목록  
      ArrayList<Shape> arShape = new ArrayList<Shape>();  
      Random Rnd = new Random();  
      Activity mParent;

      public GameView(Context context) {  
        super(context);  
        mParent = (Activity)context;

        // 빈 화면으로 시작하고 잠시 후에 게임 시작  
        status = BLANK;  
        mHandler.sendEmptyMessageDelayed(0,DELAY);  
      }

      public void onDraw(Canvas canvas) {

        // 검정색 배경으로 지운다. 빈 화면이면 지우기만 하고 리턴   
        canvas.drawColor(Color.BLACK);  
        if (status == BLANK) {  
          return;  
      }

      // 도형 목록을 순회하면서 도형 정보대로 출력한다.  
      int idx;  
      for (idx = 0; idx < arShape.size(); idx ++) {  
        Paint Pnt = new Paint();  
        Pnt.setColor(arShape.get(idx).color);

        Rect rt = arShape.get(idx).rt;  
        switch (arShape.get(idx).what) {  
          case Shape.RECT:  
            canvas.drawRect(rt, Pnt);  
            break;  
          case Shape.CIRCLE:  
            canvas.drawCircle(rt.left + rt.width()/2, rt.top + rt.height()/2,  
            rt.width()/2, Pnt);  
            break;  
          case Shape.TRIANGLE:  
            Path path = new Path();  
            path.moveTo(rt.left + rt.width()/2, rt.top);  
            path.lineTo(rt.left, rt.bottom);  
            path.lineTo(rt.right, rt.bottom);  
            canvas.drawPath(path, Pnt);  
            break;  
        }  
      }  
    }

    public boolean onTouchEvent(MotionEvent event) {  
      if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
        int sel;  
        sel = FindShapeIdx((int)event.getX(), (int)event.getY());

        // 빈 바닥을 찍었으면 무시한다.  
        if (sel == -1) {  
          return true;  
        }

        // 마지막 추가된 도형을 제대로 찍었으면 다음 단계로 진행.   
        // 빈 화면 잠시 보여준 후 새 도형 추가  
        if (sel == arShape.size()-1) {  
          status = BLANK;  
          invalidate();  
          mHandler.sendEmptyMessageDelayed(0,DELAY);

          // 엉뚱한 도형을 찍었으면 질문 후 게임 종료 또는 재시작  
        } else {  
          new AlertDialog.Builder(getContext())  
          .setMessage("재미있지! 또 할래?")  
          .setTitle("게임 끝")  
          .setPositiveButton("함더", new DialogInterface.OnClickListener() {  
            public void onClick(DialogInterface dialog, int whichButton) {  
              arShape.clear();  
              status = BLANK;  
              invalidate();  
              mHandler.sendEmptyMessageDelayed(0,DELAY);  
            }  
          })  
          .setNegativeButton("안해", new DialogInterface.OnClickListener() {  
            public void onClick(DialogInterface dialog, int whichButton) {  
              mParent.finish();  
            }  
          })  
          .show();  
        }  
        return true;  
      }  
      return false;  
    }

    // 새로운 도형을 목록에 추가한다.  
    void AddNewShape() {  
      Shape shape = new Shape();  
      int idx;  
      boolean bFindIntersect;  
      Rect rt = new Rect();

      // 기존 도형과 겹치지 않는 새 위치를 찾는다.  
      for (;;) {  
        // 크기는 32, 48, 64 중 하나 선택  
        int Size = 32 + 16 \* Rnd.nextInt(2);

         // 위치는 난수로 선택  
         rt.left = Rnd.nextInt(getWidth());  
         rt.top = Rnd.nextInt(getHeight());  
         rt.right = rt.left + Size;  
         rt.bottom = rt.top + Size;

         // 화면을 벗어나면 안된다.  
         if (rt.right > getWidth() || rt.bottom > getHeight()) {  
           continue;  
         }

         // 기존 도형 순회하며 겹치는지 조사한다.  
         bFindIntersect = false;  
         for (idx = 0; idx < arShape.size(); idx ++) {  
           if (rt.intersect(arShape.get(idx).rt) == true) {  
             bFindIntersect = true;  
           }  
         }

         // 겹치지 않을 때 확정한다. 겹치면 계속 새 위치 선정한다.  
         if (bFindIntersect == false) {  
           break;  
         }  
       }

       // 새 도형 정보 작성. 모양, 색상 등을 난수 선택한다.  
       shape.what = Rnd.nextInt(3);

       switch (Rnd.nextInt(5)) {  
         case 0:  
           shape.color = Color.WHITE;  
           break;  
         case 1:  
           shape.color = Color.RED;  
           break;  
         case 2:  
           shape.color = Color.GREEN;  
           break;  
         case 3:  
           shape.color = Color.BLUE;  
           break;  
         case 4:  
           shape.color = Color.YELLOW;  
           break;  
       }

       shape.rt = rt;  
       arShape.add(shape);  
     }

      // x, y 위치의 도형 번호를 찾는다. 도형이 없는 위치면 -1 리턴  
      int FindShapeIdx(int x, int y) {  
        int idx;

        for (idx = 0; idx < arShape.size(); idx ++) {  
          if (arShape.get(idx).rt.contains(x, y)) {  
            return idx;  
          }  
        }  
        return -1;  
      }

      // 새 도형을 추가하고 화면을 다시 그린다. 시간 딜레이를 주기 위해 핸들러를 사용했다.  
      Handler mHandler = new Handler() {  
        public void handleMessage(Message msg) {  
          AddNewShape();  
          status = PLAY;  
          invalidate();

          String title;  
          title = "MemoryPower - " + arShape.size() + " 단계";  
          mParent.setTitle(title);  
        }  
      };  
    }  
  }

  200줄이 조금 넘는 길이로, 대부분 앞에서 배운 내용이다.

  아직 소개하지 않은 것은 가벼운 질문 대화상자 정도 뿐이다.

  액티비티는 게음 뷰를 생성하여 화면에 배치할 뿐, 아무 기능이 없다.

  실질적인 게임 운영은 전적으로 뷰가 전담하며, 액티비티는 그야말로 뷰를 담는 껍데기에 불과하다.

  Shape 클래스는 화면에 나타나는 도형 하나를 표현한다.

  사각형·원·삼각형, 세 가지 종료의 모양을 가지며, 임의의 색상을 가질 수 있다.

  도형 정보만을 가지므로 메서드는 따로 없다.

  게임을 운영해주는 주체인 GameView는 게임 화면을 직접 그리기 위해 onDraw를 재정의해야 하므로 View로부터 상속받았다.

  뷰의 멤버를 보자. 상수나 난수 생성기, 부모 액티비티 참조를 제외하고 게임 운영에 사용되는 주요 필드는 두 개밖에 없다.

  arShapre는 Shape의 배열이며 생성된 모든 도형의 정보를 저장하는 컬렉션이다.

  status는 게임의 현재 상태인데, BLANK·PLAY 둘 중 하나의 값을 가진다.

  이름이 암시하듯, BLANK는 빈 화면 상태이고, PLAY는 도형을 보여주고 입력을 기다리는 상태이다.

  DELAY 상수는 빈 화면을 보여 줄 시간인데, 게임의 난이도를 조정하는 옵션 상수이다.

  Rnd 멤버는 난수 생성기다. 모양, 위치 등이 예측 불가능해야 하므로 난수가 필요하다.

  AddNewShape는 새로운 도형을 추가한다. 생성할 때는 다음과 같은 2가지 조건을 만족해야 한다.

  (1) 화면을 벗어나서는 안 된다.

  (2) 기존 도형과 겹쳐서는 안 된다. 왜냐하면 사용자가 도형을 클릭할 때 판별이 애매해지기 때문이다.

  뷰의 안쪽인지, 바깥인지는 좌표와 뷰의 크기를 비교하면 알 수 있다.

  기존 도형과의 겹칩 여부는 교집합이 존재하는지를 보면 된다. (위에서 intersect로 해결하였음)

  AddNewShape 메서드는 직접적으로 바로 호출되지 않으며, 핸들러에서 일정시간 경과 후에 호출하도록 되어 있다.

  sendEmptyMessageDelayed 메서드로 핸들러를 호출하면, 지연 시간을 줄일 수 있으며 지연되는 동안 status는 BLANK 상태이다.

  이외에도 많은 설명이 있지만, 이것을 일일이 설명하는 것은 여러분의 실력에 도움이 되지 않을 뿐더러

  이미 너무나 길어진 지면 탓에 여러분이 흥미를 잃을 수도 있다.

  시간이 충분하다면 비슷한 난이도의 게임을 직접 만들어 보는 것도 권할만하다.

  굳이 새로운 아이디어를 낼 필요 없이, 이미 발표된 게임의 모작을 만들어 보아라.

///////////////////////////////////////////////////////////////////////////////////////

**<터치 입력>**

  이벤트를 처리하는 여러가지 방법을 알아보았으니, 이제 개별 이벤트들에 대해 알아보자.

  모바일 환경에서 가장 흔하게 발생하는 이벤트는 손가락이나 스타일러스 화면을 누르는 터치 동작이다.

  이 이벤트는 다음 콜백 메서드나 리스너의 핸들러로 받는다.

  boolean onTouchEvent (MotionEvent event) : 콜백 메서드

  boolean onTouch (View v, MotionEvent event) : 리스너, 여러 대상에 등록이 가능하므로 이벤트 대상인 v를 전달 받는다.

**MotionEvent**

  MotionEvent.getAction 메서드는 사용자가 화면에 대고 무슨 짓을 했는지에 대한 정보를 전달한다.

  ACTION\_DOWN : 화면을 눌렀다.

  ACTION\_MOVE : 누른채로 움직였다.

  ACTION\_UP : 화면에서 손가락을 뗐다.

  처음 누를 때, 누른 상태로 이동할 때, 떨어질 때 각각 다른 이벤트가 발생한다.

  현재 화면의 좌표는 MotionEvent의 getX, getY 메서드로 조사할 수 있다.

  다음은 손가락으로 화면을 터치하여, 자유곡선을 그리는 예제이다.

  GUI 환경에서 가장 먼저 만들어 보는, 아주 기본적인 예제이다.

  이벤트를 처리하고 내부적으로 자료를 관리하기 때문에 길이는 조금 길다.

**Input/FreeLine.java**

  public class FreeLine extends Activity {  
    private MyView vw;  
    ArrayList<Vertex> arVertex;  
   
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        vw = new MyView(this);  
        setContentView(vw);

**arVertex = new ArrayList<Vertex>();**  
    }

    // 정점 하나에 대한 정보를 가지는 클래스  
    public class Vertex {  
      Vertex(float ax, float ay, boolean ad) {  
        x = ax;  
        y = ay;  
        Draw = ad;  
      }  
      float x;  
      float y;  
      boolean Draw;  
    }  
   
    protected class MyView extends View {  
      Paint mPaint;

      public MyView(Context context) {  
        super(context);  
           
        // Paint 객체 미리 초기화  
        mPaint = new Paint();  
        mPaint.setColor(Color.BLACK);  
        mPaint.setStrokeWidth(3);  
        mPaint.setAntiAlias(true);  
      }

      public void onDraw(Canvas canvas) {  
        canvas.drawColor(0xffe0e0e0);  
     
        // 정점을 순회하면서 선분으로 잇는다.  
        for (int i=0;i<arVertex.size();i++) {  
          if (arVertex.get(i).Draw) {  
            canvas.drawLine(arVertex.get(i-1).x, arVertex.get(i-1).y,   
            arVertex.get(i).x, arVertex.get(i).y, mPaint);  
          }  
        }  
      }

      // 터치 이동시마다 정점들을 추가한다.  
      public boolean onTouchEvent(MotionEvent event) {  
        if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
          arVertex.add(new Vertex(event.getX(), event.getY(), false));  
          return true;  
        }  
        if (event.getAction() == MotionEvent.ACTION\_MOVE) {  
          arVertex.add(new Vertex(event.getX(), event.getY(), true));  
          **invalidate();** // 새로 추가된 선을 그리기 위해 무효화하고, 간접적으로 onDraw를 호출하는 역할  
          return true;  
        }  
        return false;  
      }  
    }  
  }

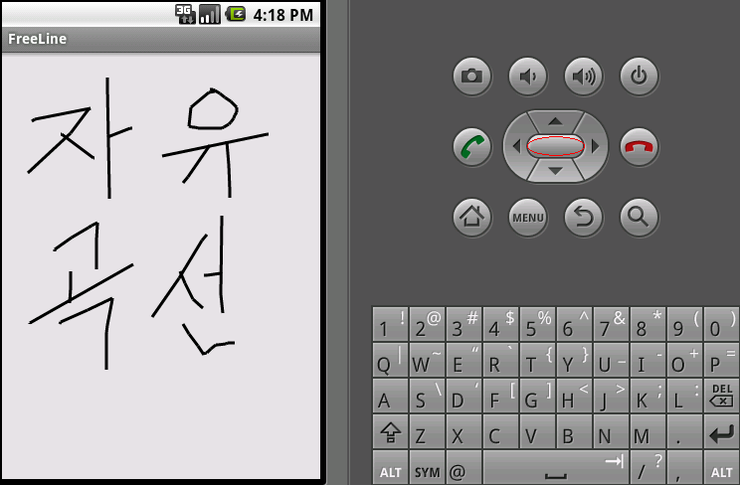
  Vertex 클래스는 곡선이 지나가는 궤적의 한 점을 나타낸다.

  x·y 멤버는 점의 좌표이며, 이 점을 순서대로 연결하면 곡선이 된다.

  Vertex의 Draw 멤버는 다음 좌표까지 선을 그을 것인지 아니면 이동만 할 것인지를 기억한다.

  곡선을 구성하는 점의 개수는 제한이 없으므로, 고정 크기의 배열을 사용해서는 안되며 반드시 가변 크기를 지원하는 배열이어야 한다.

  추가만 되고, 중간에 삽입, 삭제할 일은 없으므로 배열 형식의 ArrayList가 가장 적합하다. (모른다면 Java Collection을 공부하라)



**돌발문제**

  만약 여기서 특정 키를 눌렀을 때, 예를 들어 빨간색으로 칠한 키패드 중앙버튼처럼

  기존에 그렸던 그림이 모두 지워지려면 어떻게 해야 하는가?

  한 번 생각해보기 바란다. 일단 이 부분은 들어가야 한다.

  이 setFocusable에 대해서는 밑에서 잠시 뒤에 만나보도록 하자.

  vw = new MyView(this);  
  **vw.setFocusable(true);**

**vw.setFocusableInTouchMode(true);**  setContentView(vw)

  해답은 마지막에 보기로 하자. 문제 또한 키보드 입력까지 읽고 풀도록 하자.

**<키보드 입력>**

  모바일 장비도 입력을 위해 키보드가 필요하다.

  장비에 따라 하드웨어 키보드를 가진 것도 있고, 스크린 키보드를 가진 것도 있다.

  에뮬레이터에서는 이 둘 모두 제공하고 있다. 키보드 이벤트는 다음 콜백 메서드가 처리한다.

**boolean onKeyDown (int keyCode, KeyEvent event)**

  뷰를 상속받았다면, 이 메서드를 재정의하여 키 입력을 처리하고

  그렇지 않다면 **View.OnKeyListener** 인터페이스의 **onKey** 메서드를 구현한 후 리스너로 등록한다.

**boolean onKey (View v, int keyCode, KeyEvent event)**

  keyCode는 사용자가 누른 키의 식별자며, 다음과 같은 상수가 정의되어 있다.

  키의 개수가 굉장히 많으므로, 전체 리스트는 레퍼런스를 참고하도록 하자.



  KeyEvent 클래스는 키보드 이벤트에 대한 여러 가지 정보를 구하는 메서드를 제공한다.

  getKeyCode 메서드는 눌러진 키의 코드를 조사하는데, 이 정보는 keyCode 인수로 따로 전달되므로 일부러 조사할 필요가 없다.

  getAction 메서드는 키보드에 어떤 동작을 했는지를 나타내는 다음 세 가지 값 중 하나를 리턴한다.



  이외에 조합키의 상태나 이벤트 발생 시간을 조사하는 메서드도 제공되므로, 필요하다면 조사할 수 있다.

  다음 예제는 키 입력을 받아 화면상의 원을 상하좌우로 이동시킨다.

  다소 식상해 보일지 모르겟지만, 그래도 완성도가 높은 예제이므로 실습을 해볼 필요가 있다.

**Input/MoveCircle.java**

  public class MoveCircle extends Activity {  
    private MyView vw;

    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      vw = new MyView(this);  
      vw.setFocusable(true);  
      vw.setFocusableInTouchMode(true);  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      float mX,mY;  
      int mColor;

      public MyView(Context context) {  
        super(context);  
        mX = 100;  
        mY = 100;  
        mColor = Color.BLUE;  
      }

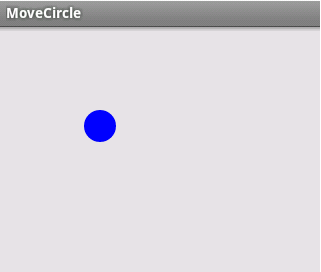
      public void onDraw(Canvas canvas) {  
        Paint Pnt = new Paint();  
        canvas.drawColor(0xffe0e0e0);  
        Pnt.setColor(mColor);  
        Pnt.setAntiAlias(true);  
        canvas.drawCircle(mX,mY,16,Pnt);  
      }

      public boolean onKeyDown(int KeyCode, KeyEvent event) {  
        super.onKeyDown(KeyCode, event);  
        if (event.getAction() == KeyEvent.ACTION\_DOWN) {  
          switch (KeyCode) {  
            case KeyEvent.KEYCODE\_DPAD\_LEFT:  
              mX-=5;  
              invalidate();  
              return true;  
            case KeyEvent.KEYCODE\_DPAD\_RIGHT:  
              mX+=5;  
              invalidate();  
              return true;  
            case KeyEvent.KEYCODE\_DPAD\_UP:  
              mY-=5;  
              invalidate();  
              return true;  
            case KeyEvent.KEYCODE\_DPAD\_DOWN:  
              mY+=5;  
              invalidate();  
              return true;  
            case KeyEvent.KEYCODE\_DPAD\_CENTER:  
              if (mColor == Color.BLUE) {  
                mColor = Color.RED;  
              } else {  
                mColor = Color.BLUE;  
              }  
              invalidate();  
              return true;  
          }  
        }  
        return false;  
      }  
    }  
  }

  액티비티는 뷰 객체를 생성해서 내용물로 채우기만 할 뿐, 특별한 코드가 없으며 대부분의 코드는 뷰에 작성되어 있다.

  단, 키 입력은 포커스를 가진 뷰에게만 전달되므로 생성된 뷰에 포커스는 반드시 주어야 한다.

  뷰의 멤버로 선언된 mX, mY는 원의 현재 좌표이며, mColor는 색상이다.



  예제를 실행한 후 키패드를 누르면, mX·mY가 각각 5씩 이동할 것이다.

  실습 컴퓨터의 키보드가 에뮬레이터의 키패드와 연결되어 있으므로 PC의 키보드를 직접 눌러도 되며, 마우스 휠을 굴려도 된다.

  또한 키패드의 중앙 버튼은 mColor를 파란색과 빨간색으로 토글(반전)한다.

**<위젯의 이벤트 처리>**

  이번에는 위젯의 이벤트 처리를 해보자. 위젯도 어차피 뷰이므로 이벤트를 받는 방법은 동일하다.

  하지만 다수의 위젯에 대한 이벤트를 한꺼번에 처리할 수 있는 좀 더 효율적인 방법이 존재한다.

  버튼의 경우 가장 흔하게 발생하는 이벤트이다.

  버튼은 단순한 차일드 위젯이며 Button 클래스를 바로 사용하는 것이 보통이므로 상속을 받지 않고도 이벤트를 처리할 수 있다.

  그래서 클릭 이벤트에 대해서는 콜백 메서드가 정의되어 있지 않으며, **반드시 리스너로 이벤트를 받아야 한다.**

**View.OnClickListener** 인터페이스를 구현하고 다음 메서드에 대한, 클릭에 대한 코드를 작성하면 된다.

**void onClick (View v)**

  터치나 키보드 입력에 비해 클릭이라는 동작이 워낙 단순하기 때문에 클릭된 뷰를 전달하는 것 외에는 별도의 인수가 없다.

  단순히 어떤 버튼이 클릭되었다는 사실만 알려줄 뿐이다. 테스트를 위한 레이아웃을 다음과 같이 작성한다.

**Input/fruit.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    android:gravity="center\_horizontal"  
    >  
  <TextView  
    android:id="@+id/fruit"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:textColor="#ffff00"  
    android:textSize="20pt"  
    android:text="과일"  
    />  
  <LinearLayout  
    android:layout\_width="wrap\_content"  
    android:layout\_height="fill\_parent"  
    >  
  <Button  
    android:id="@+id/apple"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:text="Apple"  
    />  
  <Button  
    android:id="@+id/orange"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:text="Orange"  
    />  
  </LinearLayout>  
  </LinearLayout>

  텍스트 뷰와 버튼들은 코드에서 참조해야 하므로 반드시 id를 주어야 한다.

**Input/Fruit/java**

  // 임시 객체로 핸들러 만들기  
  public class Fruit extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.input\_fruit);

      Button btnApple=(Button)findViewById(R.id.apple);  
      btnApple.**setOnClickListener**(new Button.OnClickListener() {  
        public void **onClick(View v)** {  
          TextView textFruit=(TextView)findViewById(R.id.fruit);  
          textFruit.setText("Apple");  
        }  
      });

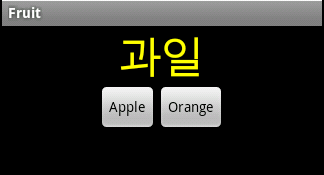
      Button btnOrange=(Button)findViewById(R.id.orange);  
      btnOrange.**setOnClickListener**(new Button.OnClickListener() {  
        public void **onClick(View v)** {  
          TextView textFruit=(TextView)findViewById(R.id.fruit);  
          textFruit.setText("Orange");  
        }  
      });  
    }  
  }

  findViewById 메서드로 XML 레이아웃에 정의된 R.id.apple 버튼 객체를 찾고

  임시 리스너 객체를 생성하여 버튼의 setOnClickListener 메서드로 등록했다.

  onClick 메서드에는 텍스트 뷰 객체를 찾아 문자열을 "Apple"로 변경하는 코드가 작성되어 있다.

  따라서 처음엔 과일이라고 텍스트를 보여주다가, 버튼을 누르면 Apple이나 Orange라고 보여줄 것이다.



  잘 동작하지만, 왠지 두 버튼의 핸들러 코드가 비슷해 다소 중복된다는 느낌을 지울수가 없다.

  또한 임시 객체가 매번 생성되므로, 메모리가 낭비되는 치명적인 문제가 있다.

  비슷한 코드가 생기면 당연히 통합하는 것이 리팩토링의 가장 기초이다.

  다행히 안드로이드는 **하나의 리스너를 여러 뷰에 대해 등록하는 것을 허용한다.**

  한 객체를 두 번 사용하려면, 이름이 필수적이므로 익명 클랙스의 임시 객체를 생성하는 방법을 쓸 수는 없다.

  다음과 같이 코드를 작성해보자. 동작은 앞 예제와 완전히 동일하다.

  // 핸들러 통합하기 : 액티비티 자체를 리스너로 구현하기

  public class Fruit extends Activity **implements View.OnClickListener** {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.input\_fruit);

      Button btnApple=(Button)findViewById(R.id.apple);

      btnApple.setOnClickListener(**this**);  
      Button btnOrange=(Button)findViewById(R.id.orange);  
      btnOrange.setOnClickListener(this);

    }

    public void onClick(View v) {  
      TextView textFruit=(TextView)findViewById(R.id.fruit);  
      switch (v.getId()) {  
        case R.id.apple:  
          textFruit.setText("Apple");  
          break;  
        case R.id.orange:  
          textFruit.setText("Orange");  
          break;  
      }  
    }  
  }

  액티비티 자체를 리스너로 만든 것이다. 우리는 앞서 이벤트 처리 6가지의 방법 중에 3번째 방법으로 소개했었다.

  액티비티가 OnClickListener 인터페이스를 직접 구현했으므로, 이 경우 리스너는 액티비티 자신인 this이다.

  굳이 설명하지 않아도, 흐름을 보면 이것이 왜 다른지에 대해 알 수 있을 것이다.

  또한 이제 콜백 메서드와 리스너의 원형이 왜 다른지에 대해서도 이해할 수 있을 것이다.

**콜백**은 특정 뷰 클래스 소속이므로 이벤트 발생에 대상이 정해져있지만,

**리스너**는 위 예제에서 보다시피 여러 위젯 (심지어 다른 종류의 위젯까지도)에 의해 공유될 수 있으므로

  대상 뷰가 누구인지를 전달받아야 하는 것이다.

  그러나 액티비티 자체를 리스너로 사용한다는 점은 다소 부담스럽다.

  최상위의 액티비티는 그대로 두고 별도의 리스너 객체를 멤버로 선언한 후, 이 멤버를 리스너로 사용하는 것이 좀 더 깔끔하다.

  // 핸들러 통합하기 : 액티비티의 멤버로 리스너 구현하기  
  public class Fruit extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.input\_fruit);

      findViewById(R.id.apple).setOnClickListener(mClickListener);  
      findViewById(R.id.orange).setOnClickListener(mClickListener);  
    }

**Button.OnClickListener mClickListener = new View.OnClickListener()** {  
      public void onClick(View v) {  
        TextView textFruit=(TextView)findViewById(R.id.fruit);  
        switch (v.getId()) {  
          case R.id.apple:  
            textFruit.setText("Apple");  
            break;  
          case R.id.orange:  
            textFruit.setText("Orange");  
            break;  
        }  
      }  
    };  
  }

  다른 방법에 비해, 버튼의 개수가 늘어나더라도 case문만 늘려주면 되므로 굉장히 편해보인다.

  이외에도 많은 방법이 있으며, 더 다양한 변형이 존재한다.

  문제를 푸는 방법이 많아 헷갈려 보이지만, 남들이 다양한 코드를 구사하므로 어쩔 수 없다.

**중요한 것은 세 가지 방법 모두 리스너 인터페이스를 구현한다는 점에서 동일하다는 것을 이해하는 것이다.** 

**<포커스 관리>**

  키보드 이벤트는 포커스를 가진 뷰에게만 전달된다.

  키보드가 하나 밖에 없기 때문에 특정 시점에서 입력을 받을 수 있는 뷰는 하나 밖에 없다.

  포커스는 다음 키 입력을 받을 뷰가 누구인지를 가리키는 것이며, 대부분 운영체제는 포커스를 가진 뷰를 색다르게 표현한다.

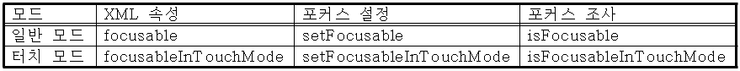


  위처럼 포커스를 가진 뷰는 특별하게 표시된다.

  그러나 모바일 장비에서 포커스는 대부분 필요하지 않다. 왜냐하면 손가락으로 직접 누르기 때문이다.

  하지만 반대로 문자를 입력하는 경우는, 터치모드라 하더라도 깜빡깜빡 입력받는 포커스를 표시해야 한다.

  따라서 안드로이드에서는 포커스를 설정할 것인지, 설정하지 않을 것인지 선택할 수 있도록 만들어져있다.



  키보드 입력을 받아야 하는 뷰는 반드시 포커스를 가져야 한다. 그래서 우리는 MoveCircle 예제에서 Focus를 설정했었다.

  다시 한번 MoveCircle 예제의 onCreate 메서드를 살펴보겠다.

  public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      vw = new MyView(this);  
      vw.**setFocusable(true);**  
      vw.**setFocusableInTouchMode(true);**  
      setContentView(vw);  
    }

  만약 이 두 메서드 호출을 생략하면, 아무리 키를 눌러도 원은 움직이질 않는다.

  그렇다면 setFocusable(true)만 설정해놓으면 어떻게 될까?

  일반 모드에서 키 입력이 잘 되는 것으로 보인다. 하지만 MoveCircle 예제에서 확인해보면

  첫 번째 키 입력이 포커스 이동 명령으로 해석하고는, 무시해 버린다.

  그래서 실제 실행해보면, 키를 두 번째 입력하고부터 움직이게 된다.

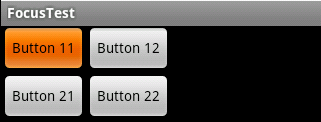
  그러므로 터치 모드에서 정상적인 동작을 위해, 2개의 포커스를 설정해야 한다.

**포커스의 이동방식**

  다음은 일반 모드에서 포커스의 이동방식을 연구해보자.

**Input/focustexst.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <TableLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:layout\_width="fill\_parent" android:layout\_height="fill\_parent">  
    <TableRow>  
      <Button android:id="@+id/btn11" android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content" android:text="Button 11" />  
      <Button android:id="@+id/btn12" android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content" android:text="Button 12" />  
    </TableRow>  
    <TableRow>  
      <Button android:id="@+id/btn21" android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content" android:text="Button 21" />  
      <Button android:id="@+id/btn22" android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content" android:text="Button 22" />  
    </TableRow>  
  </TableLayout>



  지극히 상식적인 이동방식이다. 포커스가 사각형을 벗어나지 못하고 안에서 빙빙빙 돌 것이다.

  우리가 아는 상식과는 다르다. 예를 들어 Button11에서 위를 누르면 Button21로 오고

  Button12에서 오른쪽으로 누르면, Button21로 와야하는데, 우리의 생각과는 다르다.

  이럴 때는 특정 방향으로 이동했을 때, 누가 받을지 지정할 수 있다.

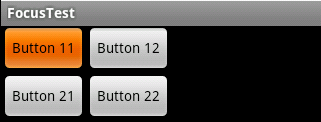


  이 속성들을 사용하여, 우리가 일반적으로 사용하는 버튼의 느낌이 나도록 설정해보자.

  이름이 명시적이어서 굳이 설명이 없어도, 알 수 있을 것이다.

  <?xml version="1.0" encoding="utf-8"?>  
  <TableLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:layout\_width="fill\_parent" android:layout\_height="fill\_parent">  
    <TableRow>  
      <Button android:id="@+id/btn11" android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content" android:text="Button 11"  
        android:nextFocusLeft="@id/btn22"   
        android:nextFocusUp="@id/btn21" />  
      <Button android:id="@+id/btn12" android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content" android:text="Button 12"   
        android:nextFocusRight="@id/btn21"   
        android:nextFocusUp="@id/btn22" />  
      </TableRow>  
      <TableRow>  
        <Button android:id="@+id/btn21" android:layout\_width="wrap\_content"  
          android:layout\_height="wrap\_content" android:text="Button 21"   
          android:nextFocusLeft="@id/btn12"   
          android:nextFocusDown="@id/btn11" />  
        <Button android:id="@+id/btn22" android:layout\_width="wrap\_content"  
          android:layout\_height="wrap\_content" android:text="Button 22"   
          android:nextFocusRight="@id/btn11"   
          android:nextFocusDown="@id/btn12" />  
      </TableRow>  
    </TableLayout>

  실행 결과는 같지만, 동작은 다를 것이다.



  Button11에서 위로 누르면, Button21로도 가고

  Button12에서 우측으로 가면, Button21로도 갈 수 있다.

  이렇게 순환의 구조를 가질 수 있도록 조절할 수 있고.

  만약 특정 뷰로 강제로 옮길 때는 원하는 뷰의 requestFocus 메서드를 호출한다.

  포커스가 변경될 때는 뷰의 onFocusChanged 이벤트가 발생하는데, 여기서 포커스 변경시에 필요한 동작을 처리한다.

  커스텀 뷰의 경우 터치 모드나 포커스 여부에 따라 배경색이나 모양을 바꿔 자신의 포커스 상태를 분명히 표시해야 하는데

  이런 처리를 onFocusChanged 이벤트 핸들러에서 하면 된다.

**<타이머 이벤트>**

  이벤트는 주로 사용자에 의해 발생하지만, 시스템의 상황 변화에 대해서도 발생한다.

  배터리가 떨어졌다거나 문자 메세지가 왔다는 것도 일종의 이벤트이며, 응용프로그램이 적절하게 처리해야 하는 대상이다.

  시간이 흘렀다는 것도 시스템의 변화 중의 하나인데, 주기적으로 어떤 작업을 해야 할 때 시간 흐름을 활용한다.

  대표적으로 시계나 타이머를 만들 때가 이에 해당하며, 애니메이션이나 백그라운드 감시 등의 작업에도 적합하다.

  일정한 주기로 반복적인 작업을 할 때는 **스레드**를 쓰는 것이 원칙적이나, 좀 더 간단하게 구현하려면 **타이머**를 사용한다.

  안드로이드는 별도의 타이머 이벤트를 제공하지 않지만, 타이머를 대체할만한 유틸리티 클래스가 몇가지 있다.

  다음 예제는 1초에 한번씩 정수 값을 증가시켜 텍스트 뷰로 출력한다.

  레이아웃에는 텍스트 뷰가 배치되어 있으며, text라는 id를 부여해두었다.

**Input/Timer.java**

**// 핸들러 멤버**  public class Timer extends Activity {  
    int value=0;  
    TextView mText;

    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.input\_timer);

      mText=(TextView)findViewById(R.id.text);  
      mHandler.**sendEmptyMessage(0);** // 이 0은 메서드 구분을 위한 것이다. 아이디라고 할 수 있다.  
    }

    Handler mHandler = new Handler() {  
      public void **handleMessage**(Message msg) { // sendEmptyMessage를 받는다.  
        value++;  
        mText.setText("Value = " + value);  
        mHandler.**sendEmptyMessageDelayed(0,1000);** // 이제 1초 뒤에 나는 0이란 메서드를 호출하겠다.  
      }  
    };  
  }

  타이머 처리를 위해 Handler 클래스를 사용했다.

  핸들러는 스레드 간의 메세지 통신을 위한 장치로써, sendMessage 메서드나 유사 메서드로 특정 핸들러에 메세지를 보낼 수 있다.

  메세지는 핸들러의 handleMessage 메서드로 전달되며, 여기서 메세지의 의미를 해석해서 필요한 처리를 한다.

  핸들러는 스레드의 경계를 넘어설 수 있는 범용적인 통신장치이며, 제대로 사용하려면 꽤 많은 것을 알아야 하지만

  간단하게 타이머 대용으로도 사용할 수 있다.

  mHandler 멤버가 바로 핸들러이며, handleMessage 메서드에서 value 값을 1 증가시켜 텍스트 뷰로 출력한다.

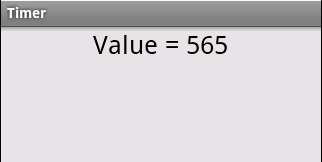
  이 핸들러는 최초 onCreate에서 sendEmptyMessage 메서드로 빈 메세지를 보냄으로서 호출한다.

  그리고 핸들러 자신이 스스로 자신에게 다시 메세지를 보내되, 이때는 sendEmptyMessageDelayed 메서드로

  1초간의 지연시간을 두어 1초후에 호출되도록 한다.

  결국 onCreate는 시동을 걸어 주기만 하면 이후는, 핸들러가 스스로를 재호출함으로써 무한히 메세지를 받게 된다.

  실행해 보면 Value 값이 1초 단위로 계속 증가할 것이다.



  핸들러가 하는 일이 그다지 복잡하지 않고, 코드가 단순하다면

  굳이 외부에 둘 필요없이 onCreate 메서드 안에서 지역적으로 초기화하는 것도 가능하다.

  다음 예제도 거의 동일하다. 다만 5초가 되면 멈추도록 설계했다.

**// onCreate에서 생성한 핸들러**  
  public class Timer extends Activity {  
    int value=0;  
    TextView mText;  
    Handler mHandler;

    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.input\_timer);

      mText=(TextView)findViewById(R.id.text);

**mHandler = new Handler()** {  
        **public void handleMessage(Message msg)** {  
          value++;  
          mText.setText("Value = " + value);  
          if (value < 5) {   
            **mHandler.sendEmptyMessageDelayed(0,1000);**  
          }  
        }  
      };  
      mHandler.sendEmptyMessage(0);  
    }  
  }

  핸들러로도 훌륭하게 타이머 흉내를 낼 수 있지만, 원래의 용도와는 조금 맞지 않은 면이 있다.

  주기적인 작업을 처리하는 좀 더 공식적이고 간편한 방법은 카운트다운 타이머 클래스를 활용하는 것이다.

  생성자는 다음과 같다.

    CountDownTimer (long millisInFuture, long countDownInterval) : 카운트 할 총 시간, 카운트 주기

  두 인수 모두, 단위는 1/1000초이다. 즉 1초를 표현하려면 1000이라고 적어야 한다.

  타이머는 총 주기 동안 실행되며, 매 주기마다 onTick 메서드를 호출하고 총 시간이 경과하면 onFinish 메서드가 호출된다.

  이제 CountDownTimer 예제에 관해 살펴보자.

  // 타이머 객체  
  public class Timer extends Activity {  
    int value=0;  
    TextView mText;  
    CountDownTimer mTimer;

    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.input\_timer);

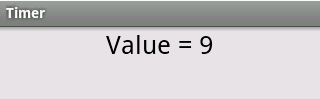
      mText=(TextView)findViewById(R.id.text);

      new CountDownTimer(10 \* 1000, 1000) { // 10초 동안, 1초를 주기로 카운트를 작동시킨다.  
        public void onTick(long millisUntilFinished) {  
          value++;  
          mText.setText("Value = " + value);  
          if (value == 5) { // 5초가 되면 종료하도록 되어있지만  
            cancel(); // 어찌된 일인지 **cancel**은 아직 작동하지 않는다.  
          }  
        }  
        public void onFinish() {  
        }  
      }.start();

    }  
  }

  0부터 카운트를 시작하므로, 9에서 멈추게 된다.

  10초까지 세고 싶다면, 11 \* 1000 혹은 11000으로 잡아야 한다.



**<이벤트 처리>**

  안드로이드에서는 이벤트를 처리하는 다양한 방법이 존재한다.

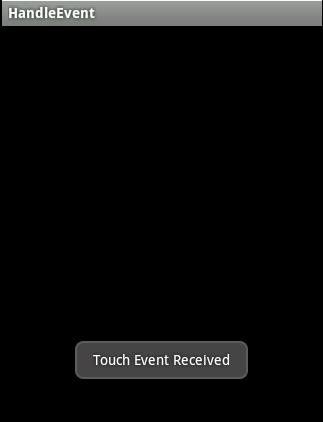
  적절한 방법을 선택해서 사용하기 위해서는 한번쯤 훓어볼 필요가 있다.

  인터페이스 구현 및 상속, 이너 클래스, 익명 클래스 등 자바의 고급 문법이 사용되므로

  잘 이해가 되지 않는다면, 자바로 돌아가 문법을 확실하게 정리할 것을 권고한다.

  이벤트를 처리하는 6가지 방법에 대해 알아볼 참이다.

  6가지 모두 실행결과는 다음과 같을 것이다.



**1. 콜백 메서드 재정의**

  콜백은 특정 이벤트 발생시 시스템에 의해 자동으로 호출되는 메서드이다.

  이 메서드에 코드를 작성해놓으면 이벤트 발생시 동작을 정의할 수 있다.

  대표적인 콜백은 다음과 같다.

  boolean onTouchEvent (MotionEvent event)

  boolean onKeyDown (int keyCode, KeyEvent event)

  boolean onKeyUp (int keyCode, KeyEvent event)

  boolean onTrackballEvent (MotionEvent event)

 사용자가 화면을 터칠할 때, 키를 누르거나 뗄 떼, 트랙볼을 굴릴 때 이 메서드들이 호출된다.

  발생한 이벤트에 대한 상세정보가 전달되는데, 예를 들어 onKeyDown으로는 눌러진 키에 대한 정보

  onTouchEvent로는 화면에 눌러진 좌표 등이 전달된다.

  콜백을 재정의하면 특정 이벤트가 발생한 시점을 정확하게 가로챌 수 있을 뿐만 아니라, 이벤트에 대한 상세정보까지 얻을 수 있다.

  일단은 인수로 전달되는 정보는 별도로 분석하지 않고, 이벤트를 받았음을 테스트를 하기 위해 토스트를 열어보기로 하자.

**Input/HandleEvent.java**

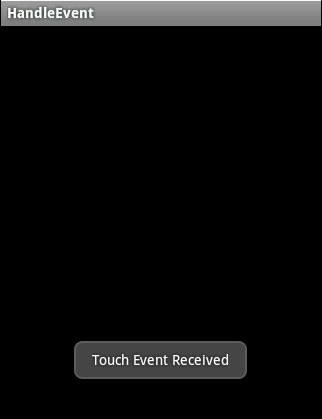
  //\* 1.핸들러 메소드 재정의 - 상속을 받아야만 재정의 가능하다.  
  public class HandleEvent extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      View vw = new MyView(this);  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

      public boolean onTouchEvent(MotionEvent event) {  
        super.onTouchEvent(event);  
        if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
          Toast.makeText(HandleEvent.this,"Touch Event Received",  
          Toast.LENGTH\_SHORT).show();  
          return true;  
        }  
        return false;  
      }  
    }  
  }

  실행하고 화면 아무 곳이나 건드리면, 이런 메세지가 출력될 것이다.

  재정의해야 할 콜백 메서드와 전달되는 인수만 다를 뿐, 키 입력이나 트랙볼 이벤트 처리 방법도 동일하다.



  특정 사건에 대해 특정 메서드를 호출하기로 프레임워크와 약속되어 있으므로, 지정된 원형대로 메서드를 재정의만 하면 된다.

  그러나 이 방법에는 몇 가지 단점 및 한계가 있다.

  (1) 메서드를 재정의 하기 위해서는 반드시 슈퍼 클래스를 상속받아야 한다.

       View의 onTouchEvent 메서드를 수정할 수 없으므로, MyView를 파생시켜야만 원하는 코드를 집어넣을 수 있다.

       뷰를 재정의하는 것은 문제가 되지 않지만, Button이나 TextView 같은 위젯의 경우는 이벤트를 처리하기 위해

       일일이 MyButton, MyTextView 클래스를 만들어야 하는 번거로움이 있다.

  (2) 프레임워크는 일반적인 이벤트에 대해 콜백 메서드를 제공하지만, 모든 이벤트에 대해 콜백이 다 정의되어 있는 것은 아니다.

       터치나 키 입력 외에도 진동 센서, 조도 센서 등 별별 이벤트가 많이 있는데 이것을 일일이 지정할 수는 없기 때문이다.

       또한 이벤트는 사용자가 정의할 수도 있으므로, 콜백 메서드는 이벤트를 처리하는 일반적인 방법이 될 수 없다.

  이런 확장성 때문에 콜백 메서드를 재정의 하는 방법은 자주 사용하는 몇 가지 이벤트에만 제한적으로 적용할 수 있으며

  그것도 반드시 상속을 해야만 쓸 수 있는 번거로움이 있다.

  그래서 안드로이드는 좀 더 범용적이고 간편한 이벤트 처리 방법을 제공하는데 그것이 바로 리스너(Listener)이다.

**2. 리스터 인터페이스 구현**

  리스너(Listener)는 특정 이벤트를 처리하는 인터페이스이다. 말 그대로 이벤트 발생 여부에 귀를 쫑긋 기울이고 있다.

  대표적인 몇 가지 리스너와 리스너에 선언된 메서드를 보자.

  View.OnTouchListener : boolean onTouch (View v, MotionEvent event)

  View.OnKeyListener : boolean onKey (View v, int keyCode, KeyEvent event)

  View.OnClickListener : void onClick (View v)

  View.OnLongClickListener : boolean onLongClick (View v)

  View.OnFocusChangeListener : void onFocusChange (View v, boolean hasFocus)

  View 클래스의 내부 인터페이스로 OnTouchListener 인터페이스가 선언되있고, 이 인터페이스는 onTouch라는 추상 메서드를 포함한다.

  이 메서드를 이벤트 핸들러라고 부른다. 다른 리스너들도 이름만 조금 다를 뿐, 거의 비슷한 구조이다.

  그러나 인터페이스는 어디까지나 껍데기일뿐, 그 자체가 이벤트를 받을 수는 없다.

  단순히 특정 이벤트를 처리하는 핸들러의 원형이 어떠해야 한다는 것을 강제할 뿐이다.

  이벤트를 처리 하는 방법은

  (1) 리스너를 구현하는 클래스를 선언하고 그 객체를 생성한다.

       클래스가 되려면 리스너의 추상 메서드인 **핸들러**를 구체적으로 구현해야 한다.

  (2) 이렇게 구현된 객체를 **리스너 객체**라고 한다.

  (3) 리스너를 구현한 다음에는 뷰에 이벤트가 발생했을 때, 핸들러가 호출되도록 연결해야 한다.

       구현만 한다고 이벤트와 리스너가 자동으로 연결되는 것은 아니므로, 어떤 리스너가 어떤 이벤트를 처리하는지 등록해야 한다.

**등록 메서드**는 리스너별로 따로 준비되어 있는데, 이름이 "set+리스너"식이어서 외우기 쉽다.

  void setOnTouchListener (View.OnTouchListener)

  void setOnKeyListener (View.OnKeyListener)

  void setOnClickListener (View.OnClickListener)

  void setOnLongClickListener (View.OnLongClickListener)

  void setOnFocusChangeListener (View.OnFocusChangeListener)

  다음 예제는 OnTouchListener 인터페이스를 구현하는 클래스를 선언한다.

  액티비티는 리스터 타입의 객체 TouchListener를 멤버로 선언 및 생성하고,

  뷰는 setOnTouchListener 메서드로 리스너 객체를 터치 이벤트와 연결했다.

  // 2.인터페이스 구현 객체 생성 - 별도의 클래스가 필요함  
  public class HandleEvent extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      View vw = new View(this);  
      // 3.리스너 등록  
      vw.setOnTouchListener(TouchListener);  
      setContentView(vw);  
    }

    // 1.리스너 구현 클래스 선언  
    class TouchListenerClass implements View.OnTouchListener {  
      public boolean onTouch(View v, MotionEvent event) {  
        if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
          Toast.makeText(HandleEvent.this,"Touch Event Received",  
          Toast.LENGTH\_SHORT).show();  
          return true;  
        }  
        return false;  
      }  
    }

    // 2.리스너 객체 생성  
    TouchListenerClass TouchListener = new TouchListenerClass();  
  }

  코드의 형식은 다르지만, 실행 결과는 앞 예제와 완벽하게 동일하다.

  리스너로 이벤트를 처리하는 절차를 다시 요약하면 이렇다.

  (1) 리스너를 구현하는 클래스를 선언하고 추상 메서드를 구현한다.

  (2) 리스너 TouchListener를 선언 및 생성한다.

  (3) 준비된 리스너 객체를 뷰의 이벤트와 연결한다.

  콜백 메서드는 상속을 받아야만 재정의 할 수 있는데, 리스너는 인터페이스이므로 임의의 클래스가 구현할 수 있다.

  그러나 인터페이스 구현을 위해 별도의 클래스를 하나 더 선언해야 한다는 것도 번거롭기는 마찬가지다.

  이 방법은 어디까지나 문법적인 설명을 위한 중간 단계일 뿐이며 좀 더 편리한 방법들이 준비되어 있다.

**3. 액티비티가 리스너 구현**

  인터페이스는 메서드 재정의를 강제할 뿐, 누가 구현하는 가는 아무런 제약이 없다.

  앞 예제처럼 별도의 클래스를 선언해도 되지만, 기존의 클래스가 인터페이스를 구현해도 아무 문제가 없다.

  선언문에서 인터페이스를 상속받고, 본체에 핸들러 메서드를 정의하기만 하면 된다.

  다음 예제는 별도의 클래스를 추가로 선언하지 않고, 액티비티가 리스너를 직접 구현한다.

  // 3.액티비티가 리스너 구현  
  public class HandleEvent **extends Activity implements View.OnTouchListener** { // 인터페이스 상속  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      View vw = new View(this);  
      **vw.setOnTouchListener(this);** // 리스너 등록  
      setContentView(vw);  
    }

**public boolean onTouch(View v, MotionEvent event)** { // 핸들러 메서드 정의  
      if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
        Toast.makeText(this,"Touch Event Received",  
        Toast.LENGTH\_SHORT).show();  
        return true;  
      }  
      return false;  
    }  
  }

  그러나 액티비티라는 큰 단위가 하위의 뷰를 위한 메서드를 제공한다는 면에서 불합리하다.

  만약 액티비티 안에 뷰가 열개 쯤 된다면, 액티비티의 구현 코드가 심하게 지저분해질 것이다.

  또한 이렇게 만들어진 뷰는 액티비티에 강하게 종속된다는 면에서 기동성이 떨어진다.

  뷰와 관련된 메서드가 뷰 자신에게 포함되어 있지 않고, 부모가 구현을 해주기 때문에 독립성이 떨어지는 것이다.

**4. 뷰가 리스너 구현**

  액티비티가 리스너를 구현할 수 있는 것처럼, 뷰도 마찬가지다.

  액티비티나 뷰, 둘 다 클래스이므로 임의의 인터페이스를 구현할 수 있다.

  다음 예제는 앞 예제와 거의 같은 형식이되, 인터페이스 구현 주체만 다르다.

  // 4.뷰에서 리스너 구현  
  public class HandleEvent extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      vw.setOnTouchListener(vw);  
      setContentView(vw);  
    }

    protected class MyView extends View implements View.OnTouchListener {  
      public MyView(Context context) {  
        super(context);  
      }

      public boolean onTouch(View v, MotionEvent event) {  
        if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
          Toast.makeText(HandleEvent.this,"Touch Event Received",  
          Toast.LENGTH\_SHORT).show();  
          return true;  
        }  
        return false;  
      }  
    }   
  }

  이벤트를 처리하는 메서드를 내부에 포함한다는 면에서 구조상 깔끔하고 뷰를 재활용하기도 유리하다.

  만약 재활용하고 싶다면, 클래스 선언문과 뷰 생성문을 통째로 가져가면 된다.

  어떤 이유로든 뷰를 상속받았다면 사실 제일 간단하고 구조적이다.

  그러나 리스너 구현만을 위해 일부러 상속을 받는다는 것은 번거로우며, (1)번 방법과 마찬가지의 문제가 있다.

**5. 익명 이너 클래스 사용**

  (3), (4)번 방법은 간편하기는 하지만, 인터페이스를 구현해 줄 적당한 클래스가 반드시 있어야 하므로, 일반적인 해법은 아니다.

  원론적으로는 (2)번 방법이 문법적으로 가장 완벽하므로, 다시 (2)번 방법으로 되돌아가보자.

  이 방법의 문제는 리스너 하나를 위해 클래스를 일일이 선언하기가 번거롭다는 점이다.

  자바의 이벤트 처리 특성상 이벤트 하나당 객체 하나씩을 만들어야 하는데 이는 너무 번거로운 일이다.

  이벤트 처리를 위해 꼭 필요한 알맹이는 사실 핸들러 메서드뿐이다.

  그런데 (2)번에서 이 코드를 핸들러로 감싸기 위해 코드가 9줄이나 더 필요하다.

  자바는 이런 경우를 위해 **익명 이너 클래스**라는 문법을 제공한다.

  (1) 상위 클래스나 인터페이시의 메서드 하나를 재정의하기 위해 클래스를 선언하는 경우

  (2) 그리고 그 클래스의 객체가 단 하나만 필요한 경우는

  굳이 클래스를 선언할 필요 없이 상속과 재정의를 동시에 할 수 있다.

  다음이 익명 이너 클래스를 사용한 예이며, (2)번 방법을 좀 더 간략화한 것이다.

  // 5.익명 이너 클래스 사용  
  public class HandleEvent extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      View vw = new View(this);  
      vw.setOnTouchListener(TouchListener);  
      setContentView(vw);  
    }

**private View.OnTouchListener TouchListener = new View.OnTouchListener()** { // 상속과 재정의를 동시에  
      public boolean onTouch(View v, MotionEvent event) {  
        if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
          Toast.makeText(HandleEvent.this,"Touch Event Received",  
          Toast.LENGTH\_SHORT).show();  
          return true;  
        }  
        return false;  
      }  
    };  
  }

  익명 이너 클래스를 정의하는 문법을 간략하게 소개하자면 다음과 같다.

  ■ 일반적인 코드                                                          ■ 축약된 코드

  class MyClass implements Interface {                           Interface obj = new Interface() {

    메서드 구현                                                                메서드 구현

  }                                                                               }**;** // 변수 선언문이기에 ';'을 붙인다

  MyClass obj = new MyClass();

  왼쪽 코드의 MyClass라는 클래스 이름이 오른쪽에는 없고, 클래스의 이름이 없으므로 이 타입의 객체는 딱 하나만 생성할 수 있다.

**6. 익명 이너 클래스의 임시 객체 사용**

  마지막으로 가장 함축된 형태의 이벤트 처리 코드를 보자.

  5번 방법은 익명 이너 클래스를 사용함으로서 코드가 대폭 짧아졌는데,

  사실 이럴 때는 객체의 이름조차도 줄 필요가 없으며, 임시 객체 하나를 생성하여 등록 메서드의 인수로 전달해 버리면 된다.

  객체 생성문 자체를 아예 setOnTouchListener 호출문 안에 넣어버리는 것이다.

  결과 코드는 다음과 같은데 처음 보면 코드가 참 이상하게 생겼다.

  // 6.익명 이너 클래스의 임시 객체 생성  
  public class HandleEvent extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      View vw = new View(this);  
      **vw.setOnTouchListener(new View.OnTouchListener()** {  
        public boolean onTouch(View v, MotionEvent event) {  
          if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
            Toast.makeText(HandleEvent.this,"Touch Event Received",  
            Toast.LENGTH\_SHORT).show();  
            return true;  
          }  
          return false;  
        }  
      });  
      setContentView(vw);  
    }  
  }

  (5)번 방법에서 (6)번 방법으로의 변화는 개념적으로 다음과 같다.

  ■ 이름있는 객체 사용                                         ■ 임시 객체 사용

  Class obj = new Class();                                   Method(new Class());

  Method(obj);

  객체에 이름을 주고 생성한 후 메서드로 전달하는 것이 아니라, 메서드 인수 목록에서 new 연산자로 바로 생성해 버린다.

  안드로이드에서는 주로 이 방법으로 이벤트 핸들러를 작성한다.

  짧고 간단해서 사용하기는 편리하지만, 언뜻 보기에도 코드가 어렵고 현란해 보인다.

  위 코드를 풀어쓰면, (5)번·(2)번 형식이 되므로 헷갈리면 문법이 압축되는 단계를 잘 연구해 보고 정리하기 바란다.

**단원 정리**

  이벤트를 처리하는 여러가지 방법에 대해 알아보았다.

  크게 상속받은 후 (1) 콜백 메서드를 재정의하는 방법과 (2) 리스너를 등록하는 방법, 2가지로 분류할 수 있다.

  그리고 리스너를 액티비티에 구현하느냐, 뷰에 구현하느냐. 주체에 따라 몇 가지 형태로 나누어지고 , 코드를 압축하는 방법이 달라진다.

  물론 이외에 문법이 허락하는 범위 내에서 또 다른 변형이 존재할 가능성도 얼마든지 있다.

**핸들러의 우선순위**

  안드로이드는 이벤트 핸들러가 중복 정의되어 있을 경우, 미리 정한 우선순위에 따라 적절한 핸들러를 선택해 호출한다.

  다음 예제로 핸들러의 우선순위를 연구해 보자.

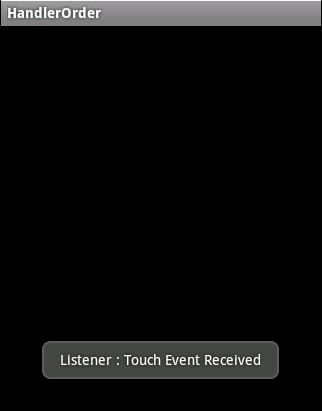
**Input/HandlerOrder/java**

  public class HandlerOrder extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      View vw = new MyView(this);  
      // 리스너 - 1순위  
      vw.setOnTouchListener(new View.OnTouchListener() {  
        public boolean onTouch(View v, MotionEvent event) {  
          if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
            Toast.makeText(HandlerOrder.this,"Listener : Touch Event Received",  
            Toast.LENGTH\_SHORT).show();  
            return true;  
          }  
          return false;  
        }  
      });  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

      // 뷰의 콜백 메서드 - 2순위  
      public boolean onTouchEvent(MotionEvent event) {  
        if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
          Toast.makeText(HandlerOrder.this,"View : Touch Event Received",  
          Toast.LENGTH\_SHORT).show();  
          return true;  
        }  
        return false;  
      }  
    }

    // 액티비티의 콜백 메서드 - 3순위  
    public boolean onTouchEvent(MotionEvent event) {  
      if (event.getAction() == MotionEvent.ACTION\_DOWN) {  
        Toast.makeText(HandlerOrder.this,"Activity : Touch Event Received",  
        Toast.LENGTH\_SHORT).show();  
        return true;  
      }  
      return false;  
    }  
  }

  보다시피 리스너가 가장 우선순위로 호출되는 것을 확인할 수 있다.  
  
  

/////////////////////////////////////////////////////////////////////////////////////

**<Toast>**

  안드로이드가 시스템 차원에서 제공하는 작은 팝업 대화상자이다.

  사용자에게 임시적인 알림 사항을 전달할 때 유용하다.

**특징**

  플로팅 형태로 화면 하단에 잠시 동안 나타나고, 일정 시간 뒤 사라진다.

  알림 사항을 전달만 하고, 입력을 받을 수 없기 때문에 사용자의 작업을 일체 방해하지 않는다.

**생성**

  static Toast makeText(Context context, int resld, int duration)

  static Toast makeText(Context context, int CharSequencem int duration)

**파라미터 설명**

  Context context : 간단하다. class명을 적어라. class명.this

  int resId : 리소스 ID이다. 따로 메세지를 입력해둔 곳이 있다면 불러오면 된다.

  int CharSequencem : 출력할 메세지를 입력한다. " " 안에 문자열을 입력하던가. 아니면 문자열의 변수명을 입력하라.

  int duration : 출력 시간 조정, LENGTH\_SHORT은 2초. LENGTH\_LONG은 4초다.

**메서드 설명**

  void setGravity (int Gravity, int xOffset, int yOffset) : 위치 설정

  void setMargin (float horizontalMargin, float verticalMargin) : 여백 설정

  void setText (CharSequence s) : 표시할 텍스트 (" "로 감싸거나, 문자열 변수를 부른다)

  void setDuration(int duration) : 시간 설정 (LENGTH\_SHORT : 2초, LENGTH\_LONG : 4초)

  void setView(View view) :  뷰 리소스 설정 (메세지 이외에 좀 더 복잡한 정보를 출력하고 싶을 때 사용)

  void show() : 메세지를 보여줌

  void cancel() : 메세지를 취소, 보이지 않게 됨 (그러나 지속시간은 여전히 유효하다)

  모든 옵션이 완료되면 최종적으로 show 메서드를 호출하여 화면에 출력

  makeText 메서드가 Toast 자체를 리턴하므로 makeText 다음에 연이어 show를 호출하는 것이 보통이다.

  한마디로 Toast / makeText / show를 이어서 적게 되면, 한 줄에 출력할 수 있다.

**예제**

  먼저 레이아웃을 보자.

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    >  
  <Button  
    android:id="@+id/shortmsg"   
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:text="짧은 메시지"  
    />  
  <Button  
    android:id="@+id/longmsg"   
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:text="긴 메시지"  
    />  
  <Button  
    android:id="@+id/count1"   
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:text="카운트 연속 출력"  
    />  
  <Button  
    android:id="@+id/count2"   
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:text="카운트 연속 출력2"  
    />  
  <Button  
    android:id="@+id/customview"   
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:text="커스텀 뷰 표시"  
    />  
  </LinearLayout>

  특정 이벤트 발생을 위해서는 버튼이 가장 간단하므로, XML에 버튼을 만들어 놓았다.

  버튼의 종류만 다를 뿐 레이아웃들은 모두 동일하다. 소스 코드는 다음과 같다.

  public class ToastTest extends Activity {  
    Toast mToast = null;  
    int count;  
    String str;

    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.output\_toasttest);  
    
      findViewById(R.id.shortmsg).setOnClickListener(mClickListener);  
      findViewById(R.id.longmsg).setOnClickListener(mClickListener);  
      findViewById(R.id.count1).setOnClickListener(mClickListener);  
      findViewById(R.id.count2).setOnClickListener(mClickListener);  
      findViewById(R.id.customview).setOnClickListener(mClickListener);  
    }

    Button.OnClickListener mClickListener = new Button.OnClickListener() {  
      public void onClick(View v) {  
        switch (v.getId()) {  
          case R.id.shortmsg:  
            Toast.makeText(ToastTest.this, "잠시 나타나는 메시지",   
            Toast.LENGTH\_SHORT).show();  
            break;  
          case R.id.longmsg:  
            Toast.makeText(ToastTest.this, "조금 길게 나타나는 메시지",   
            Toast.LENGTH\_LONG).show();  
            break;  
          case R.id.count1:  
            str = "현재 카운트 = " + count++;  
            if (mToast != null) {  
              mToast.cancel();  
            }  
            mToast = Toast.makeText(ToastTest.this, str, Toast.LENGTH\_SHORT);  
            mToast.show();  
            break;  
          case R.id.count2:  
            str = "현재 카운트 = " + count++;  
            if (mToast == null) {   
              mToast = Toast.makeText(ToastTest.this, str, Toast.LENGTH\_SHORT);  
            } else {  
              mToast.setText(str);  
            }  
            mToast.show();  
            break;  
          case R.id.customview:  
            LinearLayout linear = (LinearLayout)View.inflate(ToastTest.this,   
            R.layout.output\_toast, null);  
            Toast t2 = new Toast(ToastTest.this);  
            t2.setView(linear);  
            t2.show();  
            break;  
        }  
      }  
    };  
  }

  레이아웃에 배치된 모든 버튼의 mClickListener 객체로 설정, 이 리스너의 onClick 메서드에서 클릭된 버튼따라 분기를 하는 간단한 구조

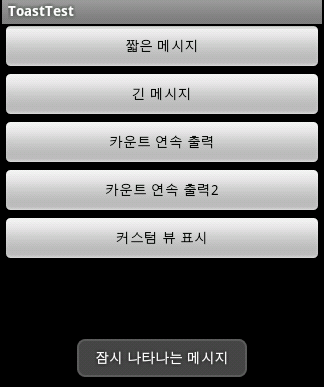
  첫번째 : SHORT, 2초간 보여준다.

  두번째 : LONG, 4초간 보여준다.

  세번째 : cancel 메서드 사용, 연속적인 출력이 힘듬.

  네번째 : setText로 대기없이 계속 출력.

  다섯번째 : 그림을 띄워준다. 그림 외에도 토스트를 띄워 복잡한 정보를 전달할 수도 있다.



  보다시피 토스트는 현재 상태를 보여준다거나, 단순한 사건 발생을 알리는 등의 단발적인 메세지에 적합하나

  그러나 일련의 메세지를 빠른 속도로 전달할 때는 적합하지 않다.

  왜냐하면 메세지가 사라지기 전에 다른 토스트가 출력되면, 현재 출력 중인 토스트가 사라질 때까지 대기한 다음 출력되기 때문이다.

  cancel 메서드를 출석하면 바로 사라지긴 합니다.

  다만 사라질 뿐, 이전 메세지의 출력 시간이 끝나야 나타나기 때문에 메세지를 다른 내용으로 즉시 변경할 수 없다.

  결국 대기 시간은 모든 토스트의 총 지속 시간의 합이며, 여러 개의 메세지를 신속하게 바꾸는데는 적합하지 않다.

  꼭 메세지를 바꿔 가며 보여주고 싶다면, 토스트를 cancel 하지 말고 setText 메서드로 메세지 내용만 변경해야 한다.

  네 번째 버튼을 누르면 카운트가 제대로 증가할 것이며, 아무리 많이 눌러도 지속 시간이 누적되어 길어지지 않는다.

  토스트가 보여주는 정보는 기본적으로 문자열이나 짧은 메세지 형태로 나타난다.

  그러나 커스텀 뷰를 띄워 좀 더 복잡한 정보를 전달할 수도 있다.

  레이아웃에 원하는 뷰를 디자인한 후 전개하여 setView 메서드로 전개된 뷰 객체를 전달하면 된다.

  setView 메서드는 레이아웃 리소스를 요구하는 것이 아니라, 뷰 객체를 요구하므로 반드시 전개한 결과를 전달해야 한다.

  물론 new 연산자로 레이아웃을 직접 생성해도 상관없다.

**마무리**

  Toast의 여러 가지 속성, 연속적인 메시지 출력 방법이나 커스텀 뷰에 대해서도 알아보았는데

  복잡해 보인다면 모두 무시하고 다음 호출문 딱 하나만 기억해 두면 된다.

   · Toast.makeText(this, "메세지", 0).show();

  학습용으로는 이 이상 복잡한 호출이 필요한 경우가 거의 없다.

**<비프음>**

  말 그대로 소리를 전달한다.

  디버깅이나 이벤트의 발생 여부, 코드의 흐름을 분석하는데 유용하다.

  (예제에 사용된 비프음은 첨부파일에 동봉되어 있다)

  MediaPlayer는 사운드와 동영상 재생 기능을 제공하는 굉장히 복잡한 객체이지만

  여기서는 간단한 브프음 출력 기능 정도만 알아볼 것이다.

  재생하고자 하는 소리 파일은 res/raw 폴더에 복사해 두어야 한다.

  하지만 마법사가 raw 폴더를 기본적으로 만들어 놓지 않으므로 폴더를 먼저 만든 후 복사해야 한다.

**정적 메서드 호출**

  static MediaPlayer create (Context context, int resid)

  인수로는 컨텍스트와 리소스 ID를 전달한다.

**재싱 및 중지**

  public void start() : 시작한다. 다만 비동기적 방식으로 동작한다. 쉽게 말해, 소리를 내면서 다른 작업도 가능하다.

  public void stop() : 정지한다.

  public void seekTo(int msec) : 특정 위치로 옮겨준다.

  public void pause() : 일시정지한다.

**Output/MessageBeep.java**

  public class MessageBeep extends Activity {  
    Beeper DingDong;  
    Beeper Ddock;

    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.output\_messagebeep);  
    
      DingDong = new Beeper(this, R.raw.dingdong); // 미리 생성한 후  
      Ddock = new Beeper(this, R.raw.ddok);

      findViewById(R.id.direct1).setOnClickListener(mClickListener);  
      findViewById(R.id.direct2).setOnClickListener(mClickListener);  
      findViewById(R.id.prepare1).setOnClickListener(mClickListener);  
      findViewById(R.id.prepare2).setOnClickListener(mClickListener);  
    }

    Button.OnClickListener mClickListener = new Button.OnClickListener() {  
      public void onClick(View v) {  
        MediaPlayer player;  
        switch (v.getId()) {  
          case R.id.direct1:  
            player = MediaPlayer.create(MessageBeep.this, R.raw.dingdong); // 호출할 때마다 객체를 생성한다.  
            player.start();  
            break;  
          case R.id.direct2:  
            player = MediaPlayer.create(MessageBeep.this, R.raw.ddok); // 그러므로 성능에 부담을 준다.   
            player.start();  
            break;  
          case R.id.prepare1:  
            DingDong.play(); // 호출만 한다.  
            break;  
          case R.id.prepare2:  
            Ddock.play(); // 그러므로 이렇게 사용하기를 권장한다.  
            break;  
        }  
      }  
    };  
  }

  class Beeper {  
    MediaPlayer player;  
    Beeper(Context context, int id) {  
      player = MediaPlayer.create(context, id);  
    }

    void play() {  
      player.seekTo(0);  
      player.start();  
    }  
  }

  위에서 표시했듯이, 소리를 2가지 방법으로 출력한다.

  첫 번째는 매번 소리를 낼 때마다 객체를 새로 생성해야 하므로, 재생 시작 속도가 느리고 반응성이 좋지 못하다.

  두 번째는 MediaPlayer 객체를 미리 생성해 놓고, 필요할 때 재생 명령만 내리는 것이다.

  하지만 재생 중에 다시 재생 명령이 들어오면, 반드시 seekTo 메서드를 호출하여 재생점을 처음으로 돌려야 한다.

  메모리를 좀 더 소모하고 미리 객체를 준비해 놓아야 한다는 점에서 편의성이 떨어지지만, 안정성이 높다.

  예제에서는 Beeper 클래스로 미디어 플레이어 객체를 감싸 미리 준비할 수 있도록 했다.

  그러므로 빠른 속도로 버튼을 누르더라도 다운되지 않을 것이다.

**SoundPool**

  사운드를 재생하는 조금 간단한 방법으로 SoundPool이 있다.

   MP3나 동영상 플레이어 같은 응용 프로그램을 만들기에는 적합하지 않으나, 단발성 사운드에는 적절하다.

**생성자**

  SoundPool(int maxStreams, int streamType, int srcQuality)

   · masStreams : 동시에 재생가능한 최대 스트림 개수

   · streamType : Audio 스트림 타입으로, 통상 STREAM\_MUSIC이다.

   · srcQuality : 샘플링 품질지정

**사운드 로드**

  int load (Context context, int resId, int priority)

  int load (String path, int priority)

  priority는 우선순의를 결정하는 것인데, 아직 효과가 없으므로 1로 주어야 한다.

**재생**

  int play (int soundID, float leftVolume, float rightVolume, int priority, int loop, float rate)

   · soundID : 재생할 사운드

   · leftVolume & rightVolume : 좌우 소리 설정. 0이 가장 작고 1이 가장 크다.

   · loop : 반복 모드. 지정 값에 +1회 반복. -1은 무한 반복

   · rate : 재생 속도. 1이 정상 속도. 0.5 절반 속도, 2가 2배속 재생이다.

**Output/SoundPoolTest.java**

  public class SoundPoolTest extends Activity {  
    SoundPool pool;  
    int ddok;

    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.output\_soundpooltest);  
    
      pool = new SoundPool(1, AudioManager.STREAM\_MUSIC, 0);  
      ddok = pool.load(this, R.raw.ddok, 1);

      findViewById(R.id.play1).setOnClickListener(mClickListener);  
      findViewById(R.id.play2).setOnClickListener(mClickListener);  
      findViewById(R.id.play3).setOnClickListener(mClickListener);  
      findViewById(R.id.play4).setOnClickListener(mClickListener);  
    }

    Button.OnClickListener mClickListener = new Button.OnClickListener() {  
      public void onClick(View v) {  
        MediaPlayer player;  
        switch (v.getId()) {  
          case R.id.play1:  
            pool.play(ddok, 1, 1, 0, 0, 1);  
            break;  
          case R.id.play2:  
            pool.play(ddok, 0.5f, 0.5f, 0, 0, 1);  
            break;  
          case R.id.play3:  
            pool.play(ddok, 1, 1, 0, 2, 1);  
            break;  
          case R.id.play4:  
            pool.play(ddok, 1, 1, 0, 0, 0.5f);  
            break;  
        }    
      }

    };  
  }

  앞 예제들과 마찬가지로 각 버튼의 클릭 핸들러에서 재생 메서드를 호출한다.

  각 버튼을 눌러 소리를 확인해 보자.



  onCreate에서 SoundPool 객체를 미리 생성해 놓고 재생할 사운드를 로드해 두었으므로 필요할 때 play 메서드만 호출하면 된다.

  재생 불륨·회수·속도 등을 섬세하게 제어할 수 있으며, 각 옵션들을 조정해서 사용하는 것도 가능하다.

**////////////////////////////////////////////////////////////////////////////////////////////////////////**

**<Bimap>**

  비록 섬세한 그리기가 가능하지만, 사진 같은 복잡한 모양은 그릴 수 없다.

  그래서 화면 장식은 미리 그려진 비트맵을 주로 많이 사용한다.

  비트맵을 사용하려면 리소스 폴더에 파일을 복사해야 한다.

  예제에 사용될 그림은 첨부파일에 있으며, 경로는 res에 drawble 폴더, mdpi에 저장하면 된다,

  파일명은 시스템의 제약으로 인해 모두 소문자로 작성해야 한다.

  리소스 이미지를 읽으려면, 먼저 리소스 관리자를 구해야 한다.

**Resources View.getResources()**

  이 객체를 읽었으면 다음 메서드로 리소스의 드로블을 읽는다.

**Drawable Resources.getDrawable(int id)**

  이 객체로부터 출력에 바로 사용할 수 있는 비트맵을 얻으려면 getBitmap 메서드를 호출해야 한다.

  결국 비트맵 하나를 읽기 위해서는 다음 세 줄을 연속으로 출력해야 한다.

**Resources r = getResources();**                                                // 리소스 관리자 얻음

**BitmapDrawable bd = (BitmapDrawable) r.getDrawable(ID);**  // 비트맵 드로블 얻음

**Bitmap bit = bd.getBitmap();**                                                    // 비트맵 추출

  이 3줄을 다음과 같이 압축할 수도 있다.

**Bitmap bit = BitmapFactory.decodeResource(getResource(), ID);**

  이와 달리 실행 중에 코드에서 비트맵을 직접 만들 수도 있는데,

  임시적인 화면 저장이나 백그라운드에서 미리 화면을 준비해 놓고자 할 때 주로 사용된다.

  코드에서 비트 맵을 생성할 때는 다음 메서드를 호출한다.

  static Bitmap createBitmap (int width, int heught, Bitmap.Config config)

  폭과 높이를 지정하고, 세 번째 인수로 색상 포맷을 전달한다.

  색상 포맷은 ALPHA\_8, ARGB\_4444, ARGB\_8888, RGB\_565가 있다.

  새로 생성된 비트맵 표면에는 아무것도 그려져 있지 않다.

  비트맵으로부터 캔버스를 생성한 후 캔버스에 그리기를 하면 이후부터 비트맵 표면에 그려진다.

  이때 캔버스의 다음 생성자를 호출한다.

  Canvas (Bitmap bitmap)

  비트맵이 곧 캔버스의 표면이 되는 셈이다.

  이 캔버스에 그려지는 그림은 메모리 내부에만 그려질 뿐, 출력 전까지 당장 눈에 보이지 않는다.

  그래서 백그라운드에 미리 그림을 그려놓을 수 있다.

  이런 용도로 사용되는 캔버스를 **오프 스크린(off screen)**이라고 부른다.

  화면에서 바로 출력되지 않고 메모리상의 캔버스를 거치는 이유는 깜박거림을 방지하기 위해서다.

  모바일 장비는 반투명 처리를 많이하며 이를 위해 배경을 먼저 출력한 후 그리는데, 바로 출력하면 깜박임이 발생하여 품질이 떨어진다.

  이를 방지하기 위해 메모리에서 다 그린 후 완성된 그림을 화면으로 보내는데, 이 기법을 **더블 버퍼링**이라고 한다.

  비트맵을 출력할 때는 Canvas의 drawBitmap 메서드를 사용한다.

  canvas.drawBitmap(Bitmap bitmap,  float left, float top, Paint paint)  
  canvas.drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint)  
  canvas.drawBitmap(Bitmap bitmap, Rect src, Rect dst, Paint paint)  
  canvas.drawBitmap(Bitmap bitmap, Matrix matrix, Paint paint)  
  canvas.drawBitmap(int []colors, int offset, int stride, int x, int y, int width, int height, boolean hasAlpha, Paint paint)

  첫 번째 메서드는 인수로 주어인 (left, top) 좌표에 비트맵 전체를 원본 그대로 출력한다.

  좌표만 지정하므로 가장 간단하지만, 확대나 축소를 지원하지 않는다.

  두번째 메서드는 src영역을 dst영역에 출력한다. 쉽게 말해 src는 내용물의 출력 지정, dst는 차일드 뷰라고 생각하라.

  src = null : 비트맵 전체 출력 / src = 사이즈 지정 : 비트맵 부분 출력 / src < dst : 비트맵 확대 / src > dst : 비트맵 축소

  원하는 부분만 원하는 크기로 출력할 수 있어서 범용적이긴 하지만, Rect 객체를 두 개나 생성해야 하므로 호출하기는 번거롭다.

  나머지 메서드는 비트맵을 약간 변형하여 출력하는 고급 출력하는 기능을 제공한다.

  public class DrawBitmap extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      setContentView(vw);  
    }

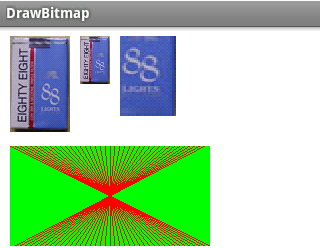
    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

      public void onDraw(Canvas canvas) {  
        Paint Pnt = new Paint();  
        canvas.drawColor(Color.WHITE);  
     
        /\* 원론적인 방법  
        Resources r=getResources();  
        BitmapDrawable bd=(BitmapDrawable)r.getDrawable(R.drawable.eighty8);  
        Bitmap bit=bd.getBitmap();  
        \*/  
     
        // 간단한 방법  
        Bitmap bit = BitmapFactory.decodeResource(getResources(), R.drawable.eighty8);  
     
        canvas.drawBitmap(bit, 10, 10, null);  
        canvas.drawBitmap(bit, null, new Rect(80,10,110,58), null);  
        canvas.drawBitmap(bit, new Rect(30,40,58,90), new Rect(120,10,120+56,10+80), null);  
     
        Bitmap BackBit = Bitmap.createBitmap(200,100,Bitmap.Config.ARGB\_8888);  
        Canvas offscreen = new Canvas(BackBit);  
        offscreen.drawColor(Color.GREEN);  
        Pnt.setColor(Color.RED);  
        for (int x=0;x<200;x+=5) {  
          offscreen.drawLine(x,0,200-x,100,Pnt);  
        }  
        canvas.drawBitmap(BackBit,10,120,null);  
      }  
    }  
  }

  제일 왼쪽은 원본 크기 그대로 출력한 것이고, 가운데는 절반으로 축소한 것이다.

  원본 사진은 60 \* 96이므로, 30 \* 48 영역에 출력하면 크기가 절반으로 줄어들 것이다.

  세번째는 원본의 일부 영역을 확대하여 출력한 것이다.



**<텍스트>**

  텍스트를 출력하는 메서드는 다음 4가지로 오버로딩 되어 있다.

  이 메서드를 통해 임의의 좌표에 문자열을 원하는 속성으로 출력할 수 있다.

  canvas.drawText(String text, float x, float y, Paint paint) : 지정 좌표에 문자열 전체 출력  
  canvas.drawText(String text, start, end, float x, float y, Paint paint) : 문자열이나 문자 배열의 일부만 출력  
  canvas.drawText(String text, int index, int count, float x, float y, Paint paint) : 문자열이나 문자 배열의 일부만 출력  
  canvas.drawTextOnPath(String text, Path path, float hOffset, float vOffset, Paint paint) : 사용자가 지정한 선을 따라 출력

  텍스트의 여러가지 속성을 변경하는 Paint의 메서드들이다.

  void setTextAlign (Paint.Align align) : 문자열의 수평 정렬. 디폴트는 왼쪽, LEFT·CENTER·RIGHT 중 하나를 선택

  void setTextSize (float textSize) : 크기

  Typeface setTypeface (Typeface typeface) : 타입 페이스

  void setUnderline Text (boolean underline Text) : 밑줄

  void setStrikeThruText (boolean strikeThruText) : 취소선

  void setSubpixelText (boolean subpixelText)

  void setTextSkewX (float skewX) : 기울임, +는 반시계 방향

  void setTextScaleX (float scaleX) : 장평

  다음 예제는 다양한 방식으로 문자열을 출력한다.

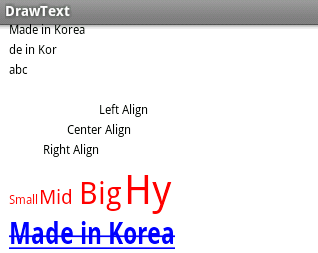
  public class DrawText extends Activity {  
   public void onCreate(Bundle savedInstanceState) {  
     super.onCreate(savedInstanceState);  
     MyView vw = new MyView(this);  
     setContentView(vw);  
   }

   protected class MyView extends View {  
     public MyView(Context context) {  
       super(context);  
     }

     public void onDraw(Canvas canvas) {  
       Paint Pnt = new Paint();  
       canvas.drawColor(Color.WHITE);  
       String str = "Made in Korea";  
       char[] arCh = {'a', 'b', 'c' };   
     
       // 기본 문자열 출력. 안티 알리아싱을 적용했다.  
       Pnt.setAntiAlias(true);  
       Pnt.setColor(Color.BLACK);  
       canvas.drawText(str, 10, 10, Pnt);  
       canvas.drawText(str, 2, 11, 10, 30, Pnt);  
       canvas.drawText(arCh, 0, 3, 10, 50, Pnt);  
      
        // 수평 정렬  
        Pnt.setTextAlign(Paint.Align.LEFT);  
        canvas.drawText("Left Align", 100, 90, Pnt);  
        Pnt.setTextAlign(Paint.Align.CENTER);  
        canvas.drawText("Center Align", 100, 110, Pnt);  
        Pnt.setTextAlign(Paint.Align.RIGHT);  
        canvas.drawText("Right Align", 100, 130, Pnt);  
     
        // 수직 정렬은 항상 글꼴의 베이스에 맞춰진다.  
        Pnt.setTextAlign(Paint.Align.LEFT);  
        Pnt.setColor(Color.RED);  
        canvas.drawText("Small", 10, 180, Pnt);  
        Pnt.setTextSize(20);  
        canvas.drawText("Mid", 40, 180, Pnt);  
        Pnt.setTextSize(30);  
        canvas.drawText("Big", 80, 180, Pnt);  
        Pnt.setTextSize(40);  
        canvas.drawText("Hy", 125, 180, Pnt);  
     
        // 여러 가지 속성 동시에 적용해 보기  
        Pnt.setColor(Color.BLUE);  
        Pnt.setTextAlign(Paint.Align.LEFT);  
        Pnt.setTextSize(30);  
        Pnt.setTypeface(Typeface.create((String)null, Typeface.BOLD));  
        Pnt.setUnderlineText(true);  
        Pnt.setStrikeThruText(true);  
        Pnt.setSubpixelText(true);  
        Pnt.setTextSkewX(-0.25f);  
        Pnt.setTextScaleX(0.8f);  
        canvas.drawText(str, 10, 220, Pnt);  
      }  
    }  
  }

  두번째 줄, Made in Korea에서 범위 지정을 하여 de in Kor를 출력하였다.

  다만 범위 지정을 할 때 주의할 것은 시작점을 포함, 끝점은 제외한다.



  drawText의 y좌표는 항상 문자열의 아래쪽 좌표이다. 그래서 y좌표를 0으로 전달하면 화면 위쪽으로 출려되어 보이지 않는다.

  Made in Korea를 출력하는 좌표가 (10,0)이 아니라 (10,10)임을 유의하자.

  글자의 모양을 결정하는 가장 중요한 정보는 타입페이스다.

  바탕, 굴림, 궁서 등 고유의 글꼴에 대한 이름인데 페이스에 따라 글자의 모양이 달라진다.

  페이스를 변경할 때는 Paint의 setTypeface 메서드를 사용하며, 이 메서드의 인수로 Typeface 객체를 전달한다.

  Typaface는 별도의 생성자가 없으므로 다음 정적 메서드로 생성해야 한다.

**static Typeface create (Typeface family, int style)**

**static Typeface create (String familyName, int style)**

  페이스 이름과 스타일을 인수로 전달한다. 스타일은 Normal, BOLD, ITALIC, BOLD\_ITALIC 등이 있다.

  페이스 이름은 고작 5개 밖에 안된다. 데스크톱에 비해 메모리 제약이 심하므로 다양하지 못하다.

  심지어 윈도우 모바일의 경우 오로지 하나의 폰트만 제공된다.

  각 타입페이스 별로 어떤 모양을 가지는지 확인해보자.

  public class TypeFace extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

      public void onDraw(Canvas canvas) {  
        Paint Pnt = new Paint();  
          int y = 1;  
          Pnt.setAntiAlias(true);  
          canvas.drawColor(Color.WHITE);  
          Pnt.setTextSize(30);  
          Pnt.setTypeface(Typeface.create(Typeface.DEFAULT, Typeface.NORMAL));  
          canvas.drawText("Font:Default", 10, y++ \* 40, Pnt);  
          Pnt.setTypeface(Typeface.create(Typeface.DEFAULT\_BOLD, Typeface.NORMAL));  
          canvas.drawText("Font:Default Bold", 10, y++ \* 40, Pnt);  
          Pnt.setTypeface(Typeface.create(Typeface.MONOSPACE, Typeface.NORMAL));  
          canvas.drawText("Font:Monospace", 10, y++ \* 40, Pnt);  
          Pnt.setTypeface(Typeface.create(Typeface.SANS\_SERIF, Typeface.NORMAL));  
          canvas.drawText("Font:Sans Serif", 10, y++ \* 40, Pnt);  
          Pnt.setTypeface(Typeface.create(Typeface.SERIF, Typeface.NORMAL));  
         canvas.drawText("Font:Serif", 10, y++ \* 40, Pnt);  
       }  
     }    
   }

  다섯 개나 있지만 그나마 다 비슷비슷해서 큰 특징이 없다.

  serif는 끝에 장식이 있고, monospace는 글자의 폭이 같아 소스 코드 덤프 등에 주로 사용되며, 나머지는 일반 텍스트 출력에 사용된다.



  기본적인 폰트 외에 커스텀 폰트로 사용자가 원하는 폰트를 설치해 사용할 수 있다.

  적당한 폰트 파일을 구하여 assets 폴더로 복사한 후 다음 메서드로 폰트를 읽어들인다.

  (첨부파일에 balloons.ttf를 assets 폴더에 집어넣는다)

**Typeface createFromAsset (AssetManager mgr, String path)**

  폰트 등의 대용량 리소스인 asset은 AssetManager에 의해 관리되는데 컨테스트의 getAssets 메서드로 쉽게 구할 수 있다.

  두 번째 인수로 폰트 파일의 이름을 지정하면 해당 폰트를 열어 Typeface 객체를 생성한다.

  이렇게 생성된 타입 페이스를 Paint 객체의 setTypeface 메서드로 지정해 놓으면 이후부터 이 폰트의 모양대로 출력된다.

  public class CustomFont extends Activity {  
    Typeface mFont;  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      setContentView(vw);  
    
      mFont = Typeface.createFromAsset(getAssets(),"lsans.ttf");

    }

    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

      public void onDraw(Canvas canvas) {  
        Paint Pnt = new Paint();  
        canvas.drawColor(Color.WHITE);  
        String str = "Custom Font Test";  
     
        Pnt.setAntiAlias(true);  
        Pnt.setTypeface(mFont);  
        Pnt.setTextSize(30);  
        canvas.drawText(str, 10, 40, Pnt);  
      }  
    }

  }

  응용 프로그램 스스로 폰트를 내장할 수 있으므로, 환경에 상관없이 실행 가능하다는 이점이 있다.

  다만 너무 큰 폰트를 사용하면 프로그램의 용량에 불리하므로 적당한 크기의 폰트를 골라 꼭 필요할 때만 사용해야 한다.



**<Path>**

  패스(Path)는 직선, 곡선, 다각형 등의 도형 궤적 정보를 가지는 그래픽 객체이다.

  도형의 좌표 정보만을 가지므로 그 자체는 화면에 직접적으로 보이지 않으며, 캔버스의 drawPath 메서드를 호출해야 그려진다.

  패스는 한번에 그려지므로 선의 속성이나 모양 등이 일관되게 적용되는 이점이 있고, 여러 번 재사용 할 수 있다.

  패스는 다방면으로 활용성이 높은 객체이다.

  안드로이드는 곡선을 직접 그리는 메서드가 없으므로 패스에 곡선 정보를 생성한 후 패스를 출력하는 방법을 사용해야 한다.

  벡터 그래픽 정보를 파일로 저장한다거나 네트워크로 출력할 때도 유용하며 특히 클리핑에도 활용된다.

  디폴트로 생성자로 빈 패스 객체를 만든 후 다음 메서드들로 도형을 추가한다.

  reset () : path 초기화

  moveTo (float x, float y) : (x,y)로 기준점 이동

  lineTo (float x, float y) : (x,y)까지 line 그리기

  addCircle (float x, float y, float radius, Path.Direction dir) : (x,y)에 radius만큼 원을 그리기

  addRect (float left, float top, float right, float bottom, Path.Direction dir) : 대각선 두 점을 이어 사각형 그리기

  quadTo (float x1, float y1, float x2, float y2) : (x,y)부터 (x,y)까지 곡선을 그린다. 기러기 모양과 비슷하다.

  cubicTo (float x1, float y1, float x2, float y2, float x3, float y3) : 3개의 (x,y)를 기준으로 곡선을 그린다. 번개모양 비슷하다.

  rLineTo (float x, float y) : r이 붙은 메서드는 기준점을 (0,0)으로 인식한다. 그래서 상대좌표를 적어줘야 한다.

  rCubicTo(float x1, float y1, float x2, float y2, float x3, float y3) : 마찬가지로 절대좌표가 아닌 상대좌표를 적는다.

  패스를 생성했으면 캔버스의 drawPath로 패스를 그리거나 텍스트 출력에 사용한다.

**drawPath (Path path, Paint paint)**

  예제를 통해 실제적인 사용을 확인해보자.

  public class DrawPath extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

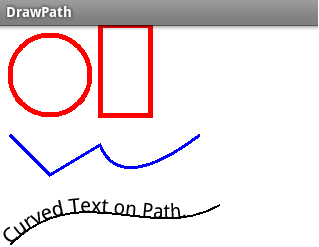
      public void onDraw(Canvas canvas) {  
        Path path = new Path();  
        canvas.drawColor(Color.WHITE);  
     
        Paint Pnt = new Paint();  
        Pnt.setStrokeWidth(5);  
        Pnt.setColor(Color.RED);  
       Pnt.setStyle(Paint.Style.STROKE);

        // 원, 사각형을 패스로 정의한 후 출력  
        path.addRect(100,00,150,90,Path.Direction.CW);  
        path.addCircle(50, 50, 40, Path.Direction.CW);  
        canvas.drawPath(path, Pnt);  
     
        // 직선 곡선을 패스로 정의한 후 출력  
        path.reset();  
        path.moveTo(10, 110);  
        path.lineTo(50, 150);  
        path.rLineTo(50, -30);  
        path.quadTo(120, 170, 200, 110);  
        Pnt.setStrokeWidth(3);  
        Pnt.setColor(Color.BLUE);  
        canvas.drawPath(path, Pnt);

        // 곡선 패스 출력  
        path.reset();  
        path.moveTo(10, 220);  
        path.cubicTo(80, 150, 150, 220, 220, 180);  
        Pnt.setStrokeWidth(2);  
        Pnt.setColor(Color.BLACK);  
        canvas.drawPath(path, Pnt);  
     
        // 곡선 패스 위에 텍스트 출력  
        Pnt.setTextSize(20);  
        Pnt.setStyle(Paint.Style.FILL);  
        Pnt.setAntiAlias(true);  
        canvas.drawTextOnPath("Curved Text on Path.", path, 0, 0, Pnt);  
      }  
    }  
  }

  원과 사각형을 한 패스에 넣어두고 한꺼번에 그렸다. 보다시피 패스는 꼭 연결되지 않은 떨어진 도형도 표현할 수 있다.

  파란색, 직선·곡선도 한번에 출력되는데 선의 모양이 복잡해지더라도 일관된 모양이 적용되는 이점이 있다.

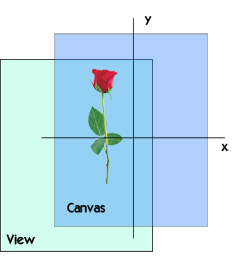
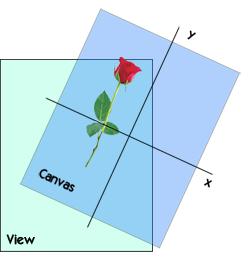


**<뷰와 캔버스의 이해>**

  뷰와 캔버스가 헷갈리시는 분들을 위해 준비했다.

  한마디로 캔버스는 뷰에 그려주는 것이다.

  그림에서 보여지듯이 View 위에 Canvas가 장미를 그려주는 것이다.

  훗날 이미지를 회전시키는 것 또한 View 자체를 돌리는 것이 아니라 Canvas를 돌리는 것이라 이해하면 되겠다.

  아무튼 View와 Canvas의 차이를 이제 이해했으리라.

**<커스텀 뷰>**

  안드로이드가 제공하는 레이아웃과 위젯만 해도 상당한 편이나, 다른 프로그램이 그렇듯 표준만으로 원하는 모든 것을 만들 수 없다.

  커스텀이란 입맛대로 만든다는 뜻이며, 모양과 기능을 마음대로 디자인 할 수 있어 상상하는 모든 것을 다 만들 수 있다.

  커스텀 위젯을 만드는 기본적인 방법은 객체 지향으 상속 기법을 활용하는 것이다.

  그러나 방법이 복잡하고 선행 지식을 많이 요구하므로, 아직 그것을 논할 단계는 아니다.

  이 장의 주제는 커스텀 위젯이 아니라 뷰의 표면에 원하는 그림을 그려내는 것이다.

  커스텀 위젯의 기본 구조는 다음과 같다.

  import android.app.\*;  
  import android.content.\*;  
  import android.graphics.\*;  
  import android.os.\*;  
  import android.view.\*;

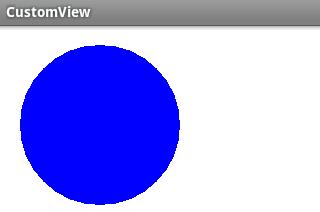
  public class CustomView extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      public MyView(Context context) {  
      super(context);  
      }

      public void onDraw(Canvas canvas) {  
        Paint Pnt = new Paint();  
        Pnt.setColor(Color.BLUE);  
        canvas.drawColor(Color.WHITE);  
        canvas.drawCircle(100,100,80,Pnt);  
      }  
    }  
  }

  View로부터 상속을 받는 MyView 클래스를 정의했는데, 이 클래스가 바로 클래스 뷰다.

  (외부 클래스로 정의해도 상관없다.)



  onDraw에 어떤 코드를 작성하는가에 따라 출력되는 모양이 달라진다.

  생각보다 자유도가 굉장히 높은 편이다. 레이아웃과 달리 픽셀 단위까지 섬세하게 지정할 수 있기 때문이다.

  이후의 예제도 구조는 동일하며 onDraw의 출력 코드만 달라진다.

**<Canvas>**

  점, 선, 원, 사각형 등 기본 도형을 그리는 메서드들이다.

  void drawPoint(x, y, paint) : (x,y) 좌표에 점을 찍는다.

  void drawLine(startX, startY, stopX, stopY, paint) : (x,y) 좌표부터 (x,y) 좌표에 선을 그린다.

  void drawCircle(x, y, radius, paint) : (x,y) 좌표에 반지름(radius)만큼 원을 그린다.

  void drawRect(left, top, right, bottom) : (left, top)와 (right, bottom)의 대각선을 기초로 사각형을 그린다.

  void drawText(text, x, y, paint) : (x,y) 좌표에 문자를 출력한다.

  void Paint.setColor(int color) : 그리기 색상

  void drawARGB(a, r, g, b) : 기존 색상에 alpha가 추가된 것. (투명 : 0xff / 불투명 : 0x00)

  void drawRGB(r, g, b) : 0~255까지 각 색상의 코드를 적으면 됨.

  void drawColor(int color) : RED, BLUE 등 원하는 색상을 적으면 된다.

  void drawPaint(paint) : 설정 값을 적용, 단순한 채우기

  void drawRoundRect(RectF r, rx, ry, paint)

   rx & ry : 모서리의 둥근 정도. 가상의 타원으로 두 값이 클수록 모서리가 둥글다.

  void drawOval(RectF r, paint) : Oval은 달걀모양, 지정한 사각형 영역에 내접한 원을 그린다.

  void drawArc(RectF r, startAngle, sweepAngle, boolean useCenter, paint)

   startAngle : 3시방향이 0도, 시계방향으로 증가

   sweepAngle : 그리는 각도 크기

   useCenter : (T) 부채꼴, (F) 호

  void drawLines(float[] pts, paint) : 배열에 저장된 선을 순서대로 그린다. 연결하고 싶으면 시작과 끝 점을 일치시킨다.

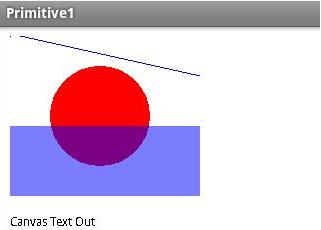
  void drawPoints(float[] pts, paint) : 배열상의 좌표를 꺼내 여러 개의 점을 찍는다.

  이제 예제를 통해 어떤 식으로 출력이 되는지 알아보자.

  public class Primitive1 extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

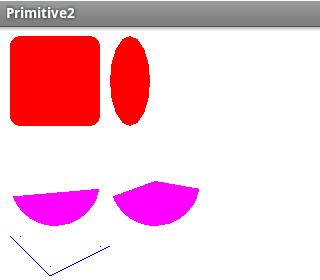
      public void onDraw(Canvas canvas) {  
        Paint Pnt = new Paint();  
        canvas.drawColor(Color.WHITE);  
        canvas.drawPoint(10,10,Pnt);  
        Pnt.setColor(Color.BLUE);  
        canvas.drawLine(20,10,200,50,Pnt);  
        Pnt.setColor(Color.RED);  
        canvas.drawCircle(100,90,50,Pnt);  
        Pnt.setColor(0x800000ff);  
        canvas.drawRect(10,100,200,170,Pnt);  
        Pnt.setColor(Color.BLACK);  
        canvas.drawText("Canvas Text Out", 10,200,Pnt);  
      }  
    }  
  }



  public class Primitive2 extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

      public void onDraw(Canvas canvas) {  
        Paint Pnt = new Paint();  
        //canvas.drawARGB(255,255,255,255);  
        //canvas.drawRGB(255,255,255);  
        //canvas.drawColor(0xffffffff);  
        Pnt.setColor(Color.WHITE);  
        canvas.drawPaint(Pnt);  
        RectF r=new RectF(10,10,100,100);  
        Pnt.setColor(Color.RED);  
        canvas.drawRoundRect(r,10,10,Pnt);  
        r.set(110,10,150,100);  
        canvas.drawOval(r,Pnt);  
        Pnt.setColor(Color.MAGENTA);  
        r.set(10,110,100,200);  
        canvas.drawArc(r,10,150,false,Pnt);  
        r.set(110,110,200,200);  
        canvas.drawArc(r,10,150,true,Pnt);  
        Pnt.setColor(Color.BLUE);  
        float[] pts={10,210,50,250,50,250,110,220};  
        canvas.drawLines(pts,Pnt);  
        Pnt.setColor(Color.BLACK);  
        float[] pts2={20,210,50,240,100,220};  
        canvas.drawPoints(pts2, Pnt);  
      }  
    }  
  }



**<Paint>**

  그리기에 대한 속성 정보를 가지는 객체이다.

  void setAntiAlias(boolean ) : 경계면을 부드럽게 처리해준다.

  void setColor(int color) : 단색을 지정한다.

  void setRGB(r, g, b) : 더 더양한 색상을 지정한다.

  void setARGB(a, r, g, b) : RGB에 Alpha가 추가된 것이다. Alpha는 투명이냐 불투명이냐를 결정할 수 있다.

  void setStrokeWidth(width) : 펜 굵기

  void setStrokeCap(Paint.Cap cap) : 끝 모양

   \* BUTT : 지정한 좌표에서 선이 끝남

   \* ROUND : 둥근 모양으로 끝이 장식된다.

   \* SQUARE : 사각형 모양이되 지정된 좌표보다 조금 더 그어진다.

  void setStrokeJoin(Paint.Join join) : 모서리처럼 선분이 만나 각지는 곳의 모양

   \* MITER : 모서리를 90도 각진 형태 (Default)

   \* BEVEL : 모서리가 깍인 형태

   \* ROUND : 둥근 형태

  void setStrokeMiter(float miter) : 0 이상의 값으로 어느 정도 각도로 뾰족하게 할 것인가 (StrokeJoin의 소속)

  void setStyle(Paint.Style style) : 외곽선과 내부 중 어느 쪽을 그릴 것인가

   \* FILL : 채우기만 하며 외곽선 X   (Default)

   \* FILL\_AND\_STROKE : 채우기도 하고 외곽선도 그린다.

   \* STROKE : 채우지는 않고 외곽선만 그린다.

  void set(Paint src) : 객체끼리 대입

  void reset() : 초기화

  cf) invalidate() : 새로 canvas를 그려준다.

  마찬가지로 예제를 보며 실제적으로 어떻게 사용이 되는지 알아보자.

  public class AntiAlias extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

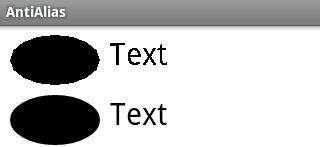
      public void onDraw(Canvas canvas) {  
        Paint Pnt = new Paint();  
        canvas.drawColor(Color.WHITE);  
        Pnt.setColor(Color.BLACK);  
        Pnt.setTextSize(30);

        // 기본 출력  
        canvas.drawOval(new RectF(10,10,100,60), Pnt);  
        canvas.drawText("Text", 110, 40, Pnt);  
       
        // 안티 알리아스 적용  
        Pnt.setAntiAlias(true);  
        canvas.drawOval(new RectF(10,70,100,120), Pnt);  
        canvas.drawText("Text", 110, 100, Pnt);  
      }  
    }  
  }

  AntiAlias는 색상 차가 뚜렷한 경계 부근에 중간색을 삽입, 도형이나 글꼴이 주변 배경과 부드럽게 어울리도록 하는 기법이다.

  사용하게 되면 출력 품질은 좋아지지만, 중간 색상을 삽입하기 위한 연산을 해야 하므로 속도는 떨어진다.

  디폴트를 false이므로 이 기법을 사용하려면, 반드시 true로 바꿔 주어야 한다.



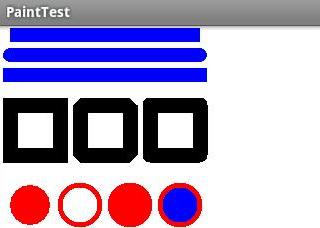
  public class PaintTest extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      MyView vw = new MyView(this);  
      setContentView(vw);  
    }

    protected class MyView extends View {  
      public MyView(Context context) {  
        super(context);  
      }

      public void onDraw(Canvas canvas) {  
      Paint Pnt = new Paint();  
      canvas.drawColor(Color.WHITE);

      // 캡 모양 테스트  
      Pnt.setColor(Color.BLUE);  
      Pnt.setStrokeWidth(14);  
      canvas.drawLine(10,10,200,10,Pnt);  
      Pnt.setStrokeCap(Paint.Cap.ROUND);  
      canvas.drawLine(10,30,200,30,Pnt);  
      Pnt.setStrokeCap(Paint.Cap.SQUARE);  
      canvas.drawLine(10,50,200,50,Pnt);

      // 조인 모양 테스트  
      Pnt.setColor(Color.BLACK);  
      Pnt.setStrokeWidth(15);  
      Pnt.setStyle(Paint.Style.STROKE);  
      Pnt.setStrokeJoin(Paint.Join.MITER);  
      canvas.drawRect(10,80,60,130,Pnt);  
      Pnt.setStrokeJoin(Paint.Join.BEVEL);  
      canvas.drawRect(80,80,130,130,Pnt);  
      Pnt.setStrokeJoin(Paint.Join.ROUND);  
      canvas.drawRect(150,80,200,130,Pnt);  
      
      // 스타일 테스트  
      Pnt.setStrokeWidth(5);  
      Pnt.setColor(Color.RED);  
      Pnt.setStyle(Paint.Style.FILL);  
      canvas.drawCircle(30,180,20,Pnt);  
      Pnt.setStyle(Paint.Style.STROKE);  
      canvas.drawCircle(80,180,20,Pnt);  
      Pnt.setStyle(Paint.Style.FILL\_AND\_STROKE);  
      canvas.drawCircle(130,180,20,Pnt);  
      Pnt.setColor(Color.BLUE);  
      canvas.drawCircle(180,180,20,Pnt);  
      Pnt.setStyle(Paint.Style.FILL);  
      Pnt.setColor(Color.RED);  
      Pnt.setStyle(Paint.Style.STROKE);  
      canvas.drawCircle(180,180,20,Pnt);  
      }  
    }  
  }



**////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////**

**<레이아웃 파라미터>**

  말이 거창하지. 사실 우리가 계속 써왔던 것들이다.

  앞에서 layout\_gravity, layout\_margin, layout\_width 등 말이다.

  어떤 사람은 이미 눈치챘는지도 모르겠다. layout이 붙은 것은 모두 부모와의 속성을 결정짓는다.

  다만 이런 것들을 '레이아웃 파라미터'라 이름짓고, 체계적으로 배운다는 것이 다르다.

  ...

  일반 속성들은 이미 겪은대로, 뷰 자체만의 속성을 지정한다. TextView 같은 일반 속성들을 떠올려보라.

  하지만 레이아웃 파라미터는 (1) 레이아웃에 소속되며, (2) 부모에게 차일드 뷰를 배치할 방법을 지시한다.

  (우리가 앞에서 사용한던 것을 멋스럽게 이름 지었을 뿐, 이미 알고 있는 것 아닌가?)

  ...

  또 다른 특징을 얘기하자면, (이미 아는 속성이다) 일반 속성은 위젯에 따라 달라진다. (TextColor는 TextView에서)

  반면 레이아웃 파라미터의 종류는 부모 레이아웃에 따라 달라진다. (이 역시 당연하다)

  이제껏 우리가 layout\_alignParentRight는 relative 속성에서만 썼던 것처럼 말이다.

  ...

  각 레이아웃별로 제공되는 레이아웃 파라미터는 다음과 같다.

|  |  |
| --- | --- |
| 레이아웃 | 파라미터 |
| ViewGroup.LayoutParams | layout\_width, layout\_height |
| ViewGroup.MarginLayoutParams | layout\_marginLeft, layout\_marginRight |
| LinearLayout.LayoutParams | layout\_gravity, layout\_weight |
| AbsoluteLayout.LayoutParams | layout\_x, layout\_y |
| RelativeLayout.LayoutParams | layout\_above, layout\_alignParentRight |

  어렵게 써놓았지만, 사실 우리가 앞에서 얘기한 배운 각각의 레이아웃과, 그 속성들을 지정하는 메서드들을 나타낸 것이다.

  맨 위의 layout\_width, layout\_height는 공통 속성이라는 것을 다 알 것이다.

  그리고 나머지는 각각의 레이아웃이 있어야만 파리미터가 적용된다.

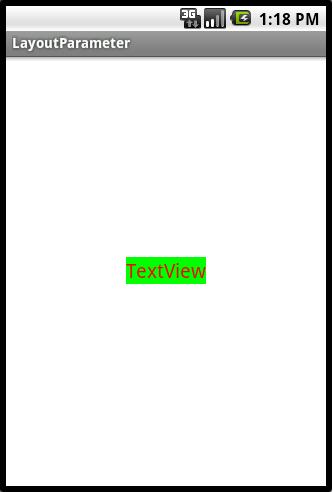
  ...

  앞서 전개와 마찬가지로 XML로 먼저 작성한 후에, 코드로 똑같이 만들어 볼 것이다.

**Layout/layoutparameter.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    android:background="#ffffff"  
    android:gravity="center"  
  >  
  <TextView    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:textColor="#ff0000"  
    android:textSize="20px"  
    android:text="TextView"  
    android:background="#00ff00"  
  />  
  </LinearLayout>

  별다른 코드 없이, 레이아웃을 전개하여 액티비티에 채우기만 한다.



  이제 이 화면을 코드로 직접 배치해보자.

  package exam.Layout;

  import android.app.\*;  
  import android.graphics.\*;  
  import android.os.\*;  
  import android.view.\*;  
  import android.widget.\*;

  public class LayoutParameter2 extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);

      LinearLayout linear = new LinearLayout(this);  
      linear.setOrientation(LinearLayout.VERTICAL);  
      linear.setGravity(Gravity.CENTER);  
      linear.setBackgroundColor(Color.WHITE);

      TextView text = new TextView(this);  
      text.setText("TextView");  
      text.setTextColor(Color.RED);  
      text.setTextSize(20);  
      text.setBackgroundColor(Color.GREEN);

**LinearLayout.LayoutParams paramtext = new LinearLayout.LayoutParams(  
     LinearLayout.LayoutParams.WRAP\_CONTENT,  
     LinearLayout.LayoutParams.WRAP\_CONTENT);  
     linear.addView(text, paramtext);**

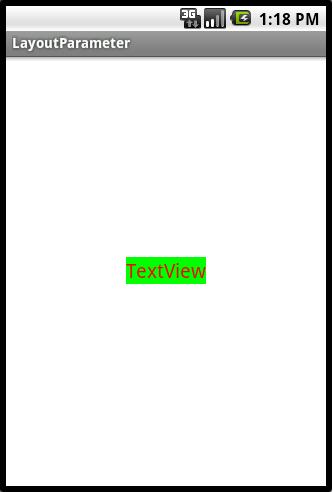
      setContentView(linear);  
    
    /\*  
     LinearLayout.LayoutParams paramlinear = new LinearLayout.LayoutParams(  
     LinearLayout.LayoutParams.FILL\_PARENT,  
     LinearLayout.LayoutParams.FILL\_PARENT);  
     setContentView(linear, paramlinear);  
    \*/

    }  
  }

  주석에 처리된 부분은 setContentView에 관한 내용이다.

  우리가 위에서 text의 레이아웃을 WRAP\_CONTENT로 지정하듯, setCotentView도 해줘야 하는데

  그렇게 하지 않은 이유는 디폴트로 저렇게 설정되어 있기 때문이다. (주석 처리 부분)



  다음은 조금 더 복잡한 경우를 보자.

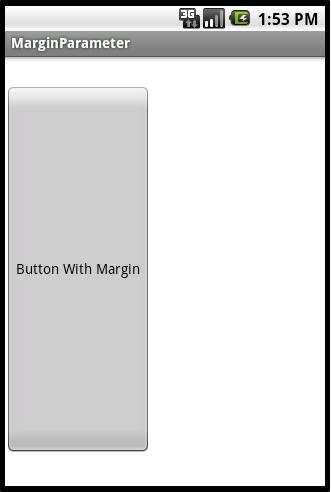
  레이아웃 파라미터가 1개 이면 그나마 좀 쉽지만, 2개 이상의 파라미터를 같이 적용할 때는 방법이 조금 달라진다.

  가장 전형적인 예는 크기와 마진을 같이 적용할 떄이다. 먼저 XML 문서부터 보자.

**Layout/marginparameter.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    android:background="#ffffff"  
  >  
  <Button    
    android:layout\_width="wrap\_content"   
    android:layout\_height="fill\_parent"   
    android:text="Button With Margin"  
    android:layout\_marginTop="30px"  
    android:layout\_marginBottom="30px"  
  />  
  </LinearLayout>

  리니어 안에 버튼을 배치했다. 크기 속성 뿐 아니라 상하에 마진도 30픽셀만큼 적용했다.



  이 화면을 코드로 생성하려면 어떻게 해야할까?

  문제는 addView의 인수로 2개 이상을 전달할 수 없다는 것이다.

  방법은 하나의 레이아웃 파라미터에 객체에 크기 정보와 마진 정보를 모두 전달해야 한다.

**Layout/MarginParameter2.java**

  package exam.Layout;

  import android.app.\*;  
  import android.graphics.\*;  
  import android.os.\*;  
  import android.view.\*;  
  import android.widget.\*;

  public class MarginParameter2 extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);

      LinearLayout linear = new LinearLayout(this);  
      linear.setOrientation(LinearLayout.VERTICAL);  
      linear.setBackgroundColor(Color.WHITE);

      Button btn = new Button(this);  
      btn.setText("Button With Margin");

      LinearLayout.LayoutParams parambtn = new LinearLayout.LayoutParams(  
      **LinearLayout.LayoutParams.WRAP\_CONTENT,  
     LinearLayout.LayoutParams.FILL\_PARENT**);  
      **parambtn.setMargins(0, 30, 0, 30);**      linear.addView(btn, parambtn);

      setContentView(linear);  
    }  
  }

  LinearLayout.LayoutParams의 생성자는 인수로 크기 정보만 받아들이므로, 생성 단계에서 마진을 적용할 수는 없다.

**<레이아웃 전개란?>**

  XML 문서에 있는 레이아웃은 생성할 뷰의 목록과 구조, 속성들을 상세하게 적어놓은 것이다.

  그러나 레이아웃은 어디까지나 뷰의 목록에 대한 정보일 뿐, 실제 뷰가 아니므로 액티비티에 놓으려면 정보대로 객체를 생성해야 한다.

  그래서 이런 XML 문서에 정의된 레이아웃과 차일드 뷰의 속성을 읽어 실제 뷰 객체를 생성해 내는 동작을 전개(Inflation)라고 한다.

  ...

  전개를 하는 핵심 메서드는 setContentView인데 XML 문서의 리소스 ID를 전달받아 이 정보대로 객체를 생성해 액티비티에 채운다.

  내부에서 어떤 동작을 할 것인가는 대충 짐작이 되는데, 이 동작을 우리가 직접 수행할 수도 있다.

  전개 과정을 연구해 보고 실제 프로젝트에 활용하기 위해 일련의 실습을 진행해 보자.

  Layout/inflation.xml

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    android:background="#ffffff"  
  >  
  <TextView    
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:gravity="center"  
    android:textColor="#ff0000"  
    android:textSize="20px"  
    android:text="TextView"  
  />  
  LinearLayout>

**Layout/Infaltion.java**

  package exam.Layout;

  import android.app.\*;  
  import android.os.\*;  
  import exam.AndroidExam.\*;

  public class Inflation extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.layout\_inflation);  
    }  
  }

  실행하면 결과는 이렇다.

  이 화면에 보이는 배치 결과가 XML 문서의 정보를 전개한 것이다.

  전개라는 것이 이름만 특별하지. 사실 우리가 계속 했던 것이다.



  그저 레이아웃의 정보대로 객체를 생성하고, 속성 변경 메서드를 순서대로 호출하는 것이다.

  따라서 이 과정을 코드로도 수행할 수 있다.

  과연 가능한지 코드만으로 직접 레이아웃을 만들어 보자.

  onCreate 메서드를 다음과 같이 수행한다.

**Layout/Inflation2.java**

  package exam.Layout;

  import android.app.\*;  
  import android.graphics.\*;  
  import android.os.\*;  
  import android.view.\*;  
  import android.widget.\*;

  public class Inflation2 extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);

      LinearLayout linear = new LinearLayout(this);  
      linear.setOrientation(LinearLayout.VERTICAL);  
      linear.setBackgroundColor(Color.WHITE);

      TextView text = new TextView(this);  
      text.setText("TextView");  
      text.setGravity(Gravity.CENTER);  
      text.setTextColor(Color.RED);  
      text.setTextSize(20);

      linear.addView(text); // 부모·자식 관계를 형성  
      setContentView(linear); // 사용자 화면에 보여준다  
    }  
  }

  먼저 new 연산자로 LinearLayout 객체를 생성했다. LinearLayout은 자바 클래스이므로 당연히 new 연산자로 생성가능하다.

  객체 생성 후 속성 값 변경 메서드를 호출하여, 방향, 배경색 등을 지정했다.

  XML 속성과 대응되는 모든 메서드가 정의되어 있으므로, XML로 가능한 모든 지정을 코드로도 수행할 수 있다.

  (자바 문법 수준의 일반적인 객체 생성 및 메서드 호출문일 뿐이다)

**<addView>**

  그러나 이 둘은 별개의 객체로 존재할 뿐이지, 아직 부모·자식 관계를 형성하지 못했다.

  텍스트 뷰를 리니어의 차일드로 집어넣어, 이 둘 사이의 관계를 정의해야 하는데 이 때는 ViewGroup의 다음 메서드를 사용한다.

  실행 중에 차일드를 추가하는데 여러 가지 버전으로 오버로딩 되어있다.

  void addView(View child)

  void addView(View child, int index)

  void addView(View child, ViewGroup.LayoutParams params)

  View 인수만을 취하는 addView 메서드는 뷰를 차일드 목록의 제일 마지막에 추가하고 디폴트 파라미터를 사용한다.

  index를 취하는 두 번째 addView 메서드는 뷰를 형제 차일드 중간에 삽입한다.

  파라미터를 취하는 세 번째 addView 메서드는 레이아웃 관련 파라미터를 인수로 전달하는데, 복잡하므로 다음에 알아보자.

**<setContentView>**

  addView까지 호출하면 리니어 안에 텍스트 뷰가 배치될 것이다.

  그러나 아직 액티비티에는 등록되지 않았으므로 사용자 눈에는 보이지 않는다.

  화면을 구성하는 단위가 액티비티이므로 뷰 그룹을 액티비티의 내용물로 등록해야 하는데, 이때는 다음 메서드를 호출한다.

  이 메서드는 세 가지 버전으로 오버로딩 되어 있다.

   · public void setContentView(int layoutResID)

   · public void setContentView(View view)

   · public void setContentView(View view, ViewGroup.LayoutParams params)

  최초 예제에서 사용한 버전은 리소스 ID를 전달하여 내부에서 전개를 하는 것이고

  이번 두 번째 예제에서 사용한 버전은 코드에서 생성한 뷰 객체를 전달한 것이다.

  ...

  액티비티의 내용물로 등록되는 것은 결국 뷰인데

  (1) 리소스로부터 뷰를 전개하여 등록할 것인가 (2) 코드에서 런타임에 생성한 뷰를 등록할 것인가가 다를 뿐이다.

  뷰를 생성하는 주체가 다를 뿐이지, 등록되는 뷰의 내용이 같다면 화면에 나타나는 결과도 같을 수 밖에 없다.

  ...

  그렇다면 리소스 ID를 인수로 받는 setContentView 메서드의 동작은 이제 대충 짐작이 될 것이다.

  하여 이 코드에서 리소스를 전개하는 부분을 직접 수행해보도록 하자.

  public class Inflation3 extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);

      /\* XML 전개를 직접 하기  
      LayoutInflater inflater = (LayoutInflater)getSystemService(  
      Context.LAYOUT\_INFLATER\_SERVICE);

      LinearLayout linear = (LinearLayout)inflater.inflate(R.layout.layout\_inflation, null);  
      setContentView(linear);  
      //\*/

      /\* 컨텍스트로부터 전개자 구하기  
      LayoutInflater inflater = LayoutInflater.from(this);  
      LinearLayout linear = (LinearLayout)inflater.inflate(R.layout.layout\_inflation, null);  
      setContentView(linear);  
      //\*/

      /\* View의 정적 메소드 사용하여 전개하기    
      LinearLayout linear = (LinearLayout)View.inflate(this, R.layout.layout\_inflation, null);  
      setContentView(linear);  
      //\*/

      //\* 가장 짧은 전개 코드  
      setContentView(View.inflate(this, R.layout.layout\_inflation, null));  
      //\*/  
    }  
  }

  전개를 하는 여러가지 방법을 기술한 것일 뿐이다. 그때 그때 편한 방법을 사용하면 된다.

  상황에 따라 전개 방법을 선택할 수 있다는 면에서 긍정적이기는 하지만 비슷비슷한 코드의 래퍼가 너무 많아 처음엔 혼란스럽다.

  똑같은 코드의 표현이므로 헷갈리지 말자.

**사용자에게 직접 전개자를 제공하는 이유**

  그렇다면 왜 사용자가 직접 전개하도록 전개자를 제공하는 것일까?

  안드로이드가 전개자를 별도로 제공하는 이유는 특정 뷰 하나만 또는 특정 뷰 그룹 하나만 따로 전개하는 것을 지원하기 위해서다.

  예를 들어, 위 예제의 TextView를 여기저기서 자주 사용한다고 해 보자.

  매번 XML 문서에 TextView 엘리먼트를 선언하기도 귀찮고, 또 선언문이나 코드의 중복이 심해지면 비효율이 발생한다.

  이럴 때 TextView 하나에 의해서만 XML 문서를 만들어 놓고 전개하여 쓸 수 있다.

  패키지 탐색기의 layout 노드 팝업 메뉴에서 [New-File]을 선택하여 mytext.xml 문서를 추가하고 다음과 같이 작성한다.

  <?xml version="1.0" encoding="utf-8"?>  
  <TextView xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:gravity="center"  
    android:textColor="#ff0000"  
    android:textSize="20px"  
    android:text="TextView"  
   />

  텍스트 뷰의 속성들을 XML 문서로 작성했다.

  예제 작성의 편의상 간단한 텍스트 뷰 하나만 작성해 봤는데, 얼마든 복잡한 뷰 그룹일 수도 있으며 다양한 속성을 미리 정의할 수 있다.

  코드에서는 다음과 같이 사용한다.

**Layout/Inflation4.java**

  package exam.Layout;

  import android.app.\*;  
  import android.graphics.\*;  
  import android.os.\*;  
  import android.view.\*;  
  import android.widget.\*;  
  import exam.AndroidExam.\*;

  public class Inflation4 extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      LinearLayout linear = new LinearLayout(this);  
      linear.setOrientation(LinearLayout.VERTICAL);  
      linear.setBackgroundColor(Color.WHITE);

      TextView text = (TextView)View.inflate(this, R.layout.layout\_mytext, null);

      linear.addView(text);  
      setContentView(linear);  
    }  
  }

  리니어만 new로 생성하고, 텍스트 뷰는 XML 문서의 정보대로 생성했다.

  전개란 XML 정보대로 뷰 하나를 생성해내는 것이므로, 임의의 뷰 객체를 XML로부터 얻을 수 있는 좋은 방법이다.

  한마디로 자주 쓰는 것은 XML로 저장해두고, 필요할 때마다 갔다가 쓰는 것이다.

  리스트 뷰나 리니어 안에 차일드 뷰를 동적으로 생성해야 하는 경우가 빈번한데

  이때 차일드 뷰를 직접 생성하는 것보다는 XML로 잘 정의해 놓고, 필요할 때마다 전개해서 사용하는 것이 훨씬 더 효율적이다.

  전개는 디자인 타임과 런타임을 연결하는 중요한 기법이므로 잘 알아두도록 하자.

FrameLayout은 형태상으로 모든 레이아웃 중에 가장 단순하다.

  차일드 뷰를 따로 배치하는 규칙이 따로 없고, 모든 차일드는 프레임의 좌상단에 나타난다.

  차일드가 2개 이상일 경우, 추가된 순서대로 겹쳐서 표시된다.

  ...

  FrameLayout은 다음의 2가지 면에서 실용성이 있다.

  ① ViewGroup의 서브 클래스로서 레이아웃의 일종이므로 실행 중에 차일드를 관리할 수 있다.

  ① addView, removeView 등의 메서드로 차일드를 추가하거나 삭제할 수 있으며 getChildCount 메서드로 차일드의 개수를

  ① 조사할 수 있다. 단순해 보여도 복수의 뷰를 담고 관리할 수 있는 엄연한 컨테이너인 것이다.

  ② 비록 겹쳐서 표시되지만 나머지는 숨기고 당장 사용할 차일드 하나만 선택적으로 나타나게 할 수 있다.

  ① 차일드의 보임 상태는 개별 뷰의 visibility 속성을 사용하여 조정하며 실행 중에도 조건에 따라 뷰의 보임 상태를 변경할 수 있다.

  ① 이 기능을 잘 활용하면 탭 형태로 화면 여러 개를 겹쳐 놓고 실행 중에 화면 스위칭이 가능하다.

  ...

  프레임 레이아웃 속성은 다음 세 가지가 있다.

  전경에 이미지를 깔아 놓을 수 있는데 그다지 실용성은 없다.

  (1) foreground : 차일드의 위쪽에 살짝 얹히는 이미지를 지정한다. 반투명한 색상을 사용할 수도 있고 이미지를 사용할 수도 있다.

  (2) foregroundGravity : 전경의 이미지를 배치할 방법을 지정한다. 디폴트는 fill이며 프레임 안에 전경이 가득 차게 확대된다.

  (2) foregroundGravity : 가운데만 전경을 둔다거나 한쪽에 밀착시켜 놓을 수 있다.

  (3) measureAllChildren : 모든 차일드의 크기를 다 측정할 것인지 아니면 보이거나 숨겨진 차일드의 크기만 적용할 것인지 지정한다.

  다음 예제는 프레임 안에 버튼과 이미지 뷰를 배치한다.

  딱 2개만 배치했는데, 차일드 개수에는 제약이 없으므로 얼마든지 많은 뷰를 포개놓을 수 있다.

  (이미지는 파일첨부 해놓았다)

**Layout/frame.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <FrameLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
      android:layout\_width="fill\_parent"  
      android:layout\_height="fill\_parent"  
      >  
  <Button  
      android:id="@+id/btn"    
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Push Button"  
      />  
  <ImageView    
      android:id="@+id/img"    
      android:layout\_width="wrap\_content"   
      android:layout\_height="wrap\_content"   
      android:src="@drawable/pride"  
      />      
  </FrameLayout>

  두 차일드 모두 visibility 속성을 따로 지정하지 않았으므로 보이는 상태이며, 따라서 2개의 뷰가 겹쳐서 나타난다.

  실무에서는 최초 보일 뷰만 보이도록 해놓고, 당장 사용하지 않을 뷰는 숨겨두는 것이 보통이다.

  버튼이 먼저 선언되었으므로 아래쪽에 깔리며, 그 위에 자동차 이미지가 포개진다.



  당장은 겹쳐서 보이지만 뷰의 보이기 상태는 실행 중에 언제든지 변경할 수 있다.

  버튼의 클릭 이벤트에 대한 리스너를 작성하여 이미지 뷰의 보이기 상태를 토글해 보자.

  아직 핸들러를 작성하는 방법은 배우지 않았으므로 그냥 구경만 해보면 된다.

**Layout/Frame.java**

  import android.app.\*;  
  import android.os.\*;  
  import android.view.\*;  
  import android.widget.\*;  
  import exam.AndroidExam.\*;

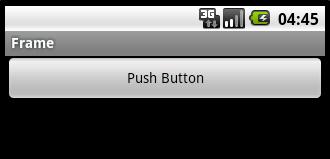
  public class Frame extends Activity {  
    public void onCreate(Bundle savedInstanceState) {  
      super.onCreate(savedInstanceState);  
      setContentView(R.layout.layout\_frame);

      Button btn = (Button)findViewById(R.id.btn);  
      btn.setOnClickListener(new Button.OnClickListener() {  
        public void onClick(View v) {  
          ImageView img=(ImageView)findViewById(R.id.img);  
          if (img.getVisibility() == View.VISIBLE) {  
            img.setVisibility(View.INVISIBLE);  
          } else {  
            img.setVisibility(View.VISIBLE);  
          }  
        }  
      });  
    }  
  }

  이미지 뷰의 보이기 상태를 조사한 후 반대로 뒤짚는 코드이다.

  버튼을 누르면 위쪽의 이미지가 숨겨지고 버튼만 남을 것이다.

  물론 버튼을 다시 누르면 이미지가 나타난다.



  간단하게 2개의 뷰를 포개 놓고 런타임에 뷰를 토글하는 예제를 만들어 봤는데

  여러 개를 겹쳐놓고 메뉴나 툴바에서 페이지를 선택하는 예가 실용적이다.

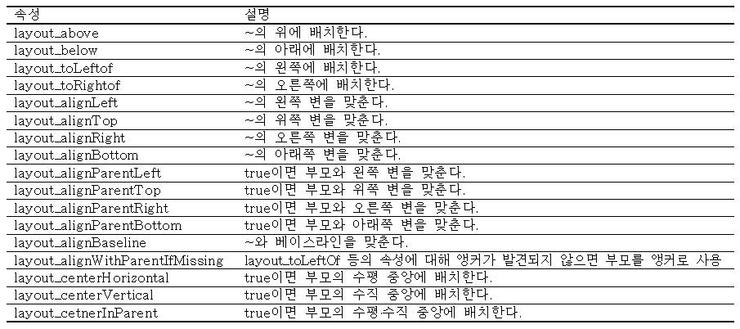
**<RelativeLayout>**

  RelativeLayout은 위젯과 부모와의 위치 관계 또는 위젯끼리의 관계를 지정함으로서, 뷰를 배치하는 레이아웃이다.

  쉽게 말해 누구 밑에 누구, 누구 옆에 누구 식으로 서로 간의 위치 관계를 지정하는 것이다.

  위젯끼리의 관계를 지정하려면 "누구"를 지칭하기 위한 id가 필요하다. 그러므로 기준이 되는 위젯에 반드시 id를 부여해야 한다.

  다음은 RelativeLayout의 주요 속성이다. 모두 레이아웃이므로 layout\_ 접두가 붙어 있다.



  리소스 컴파일러는 빠른 배치를 위해 위젯간의 관계를 한번에(one pass) 읽도록 되어 있다.

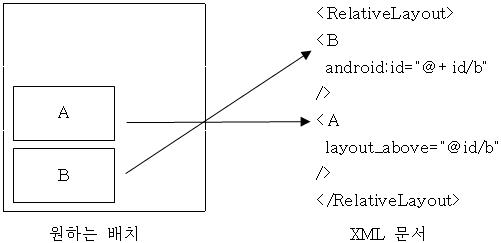
  따라서 특정 뷰가 다른 뷰의 위치에 종속적일 때, 기준이 되는 뷰를 먼저 정의해야 하며

  그러다보니 화면상의 순서와 레이아웃 상의 순서가 달라지기도 한다.

  ...

  예를 들어 A를 B의 위에 배치하고 싶다면, XML 문서상에 B를 먼저 기술한 후 A를 나중에 기술해야 한다.

  B가 정의되지도 않았는데 A를 B 위에 둔다는 지정을 할 수는 없다.



  간혹 A를 먼저 배치하고, 그 밑에 B를 배치하면 안되냐고 묻는데 당연히 안된다.

  A의 위치를 어떻게 기술하겠다는 것인가?

  어렵게 생각할 필요없이, 기준이 되는 위치를 먼저 기술한다고 생각하라.

  기준이 된다는 말은 표시를 명시할 수 있다는 뜻이다.

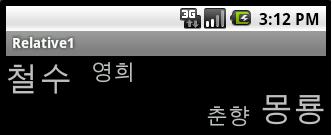
  B는 왼쪽하단이라고 설명이 되지만, A는 왼쪽 어디라고 말할 것인가?

**Layout/relative1.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
   android:layout\_width="fill\_parent"  
   android:layout\_height="wrap\_content"  
   >  
   **<TextView   
    android:id="@+id/chulsoo"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:layout\_marginRight="20px"  
    android:textSize="15pt"  
    android:text="철수"  
    />  
   <TextView  
    android:id="@+id/younghee"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:layout\_toRightOf="@id/chulsoo"  
    android:textSize="10pt"  
    android:text="영희"  
    />  
   <TextView  
    android:id="@+id/mongryong"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:layout\_below="@id/younghee"  
    android:layout\_alignParentRight="true"  
    android:layout\_marginLeft="10px"  
    android:textSize="15pt"  
    android:text="몽룡"  
    />  
   <TextView  
    android:id="@+id/chunhyang"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:layout\_toLeftOf="@id/mongryong"  
    android:layout\_alignBottom="@id/mongryong"  
    android:textSize="10pt"  
    android:text="춘향"  
    />**  </RelativeLayout>

  실행결과를 보자.

  춘향을 참조하는 위젯은 없으므로 춘향이에게는 id를 주지 않아도 상관없다,



  당연히 기준은 무엇이 되어야겠는가? 두말할 나위 없이 철수다.

  나머지 내용은 어렵지 않으니, 소스를 보면 금방 이해할 것이다.

**Layout/namecard.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
   android:layout\_width="fill\_parent"  
   android:layout\_height="wrap\_content"  
   android:background="#ffffff"  
   android:padding="5px"  
   >  
**<ImageView    
    android:id="@+id/picture"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:layout\_alignParentLeft="true"  
    android:layout\_marginRight="5px"  
    android:src="@drawable/picman"  
    />  
   <Button    
    android:id="@+id/btndel"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:text="Del"  
    android:textSize="6pt"  
    android:layout\_below="@id/picture"  
    />  
   <TextView    
    android:id="@+id/name"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:text="홍길동"  
    android:textColor="#000000"  
    android:textSize="12pt"  
    android:layout\_alignParentTop="true"  
    android:layout\_toRightOf="@id/picture"  
    />  
   <TextView  
    android:id="@+id/call"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:text="015-123-4567"  
    android:textColor="#0000ff"  
    android:textSize="6pt"  
    android:layout\_alignParentRight="true"  
    android:layout\_alignBaseline="@id/name"  
    />  
   <TextView    
    android:id="@+id/description"    
    android:layout\_width="wrap\_content"   
    android:layout\_height="wrap\_content"   
    android:text="우리 딸래미가 처음 그린 그림이다. 누구를 그린 것인지는 알 수 없으나 머리 카락이 몇 올 없는 것으로 보아 갓 태어난 동생을 그린 것으로 추측된다. 본인도 기억이 나지 않는다 하여 확인할 방법은 없다. "  
    android:textColor="#000000"  
    android:textSize="6pt"  
    android:layout\_below="@id/name"  
    android:layout\_alignLeft="@id/name"  
    />**  
  </RelativeLayout>

  직접 이 소스를 분석해보고 만들어보자. 이미지 파일은 첨부해놓았다.

  설명보다 본인의 이해가 필요한 단원이다.



  좌상단의 이미지를 제일 먼저 배치하고, 그 아래에 삭제 버튼을 배치했다.

  이름은 부모의 위쪽 변에 붙이면서 동시에 이미지의 오른쪽에 배치했다.

  전화번호는 부모의 오른쪽에 붙이되 이름과 베이스를 맞추어 가지런히 보이도록 했다.

  마지막으로 설명 문자열은 이름 밑에 붙이되 왼쪽 변에 이름과 나란히 맞춘다.

  ...

  단순히 위치만 지정하는 것이 아니라 정렬가지도 같이 지정한다는 점을 유심히 봐야 한다.

**<AbsoluteLayout>**

  AbsoluteLayout은 의미상으로 렐러티브의 반대 속성을 가지는 레이아웃이다.

  이름 그대로 관계나 순서에 상관없이 지정한 절대 좌표에 차일드 뷰를 무식하게 배치한다.

  차일드 뷰 좌표를 layout\_x, layout\_y 속성으로 지정해 놓으면 부모의 좌상단을 기준으로 한 좌표에 뷰가 배치된다.

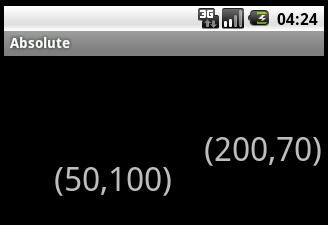
  규칙이 워낙 단순하므로 예제 하나만 만들어 보면 이해할 수 있다.

  Layout/absolute.xml

  <?xml version="1.0" encoding="utf-8"?>  
  <AbsoluteLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
      android:layout\_width="fill\_parent"  
      android:layout\_height="fill\_parent"  
      >  
  <TextView    
      android:layout\_width="wrap\_content"   
      android:layout\_height="wrap\_content"  
      android:layout\_x="50px"   
      android:layout\_y="100px"   
      android:textSize="15pt"  
      android:text="(50,100)"  
      />  
  <TextView    
      android:layout\_width="wrap\_content"   
      android:layout\_height="wrap\_content"  
      android:layout\_x="200px"   
      android:layout\_y="70px"   
      android:textSize="15pt"  
      android:text="(200,70)"  
      />  
  </AbsoluteLayout>

  좌표를 지정하여 차일드를 배치하는 방식은 지극히 상식적이고, 전통적이다.

  특별한 제약없이 임의의 위치에 뷰를 배치할 수 있어 자유도가 꽤 높은 편이다.



  그러나 장비의 해상도나 방향이 수시로 바뀔 수 있는 모바일 환경에서는 유연하지 못하고 관리하기도 어렵다.

  그래서 공식 문서에는 absolite 레이아웃을 아예 사용하지 말라고 되어 있으며, 제공되지 않는 레이아웃이라고 생각해도 무방하다.

  상용 프로그램 제작에는 사용할만한 레이아웃이 아니다.

**//////////////////////////////////////////////////////////////////////////////////////////////////////**

**베이스 정렬**

  baselineAligned 속성은 굉장히 쉽다.

  높이가 다른 차일드 뷰를 수평으로 나란히 배치할 때, 아래쪽 면을 가지런히 정렬할 것인가 아닌가를 지정한다.

  디폴트가 true이므로 문자열 높이가 달라도 보기 좋게 출력된다.

**Layout/base1.xml**

<LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
      android:orientation="horizontal"  
      android:layout\_width="fill\_parent"  
      android:layout\_height="fill\_parent"  
      **android:baselineAligned="true"**  
      >  
  <TextView    
      android:layout\_width="wrap\_content"   
      android:layout\_height="wrap\_content"   
      android:text="Medium"  
      android:textSize="10pt"  
      />  
  <TextView    
      android:layout\_width="wrap\_content"   
      android:layout\_height="wrap\_content"   
      android:text="Small"  
      android:textSize="5pt"  
      android:background="#0000ff"  
      />  
  <TextView    
      android:layout\_width="wrap\_content"   
      android:layout\_height="wrap\_content"   
      android:text="Large"  
      android:textSize="20pt"  
      android:typeface="serif"  
      />  
  </LinearLayout>



  반대로 false인 경우는 이렇게 된다.



**차일드 영역 분할**

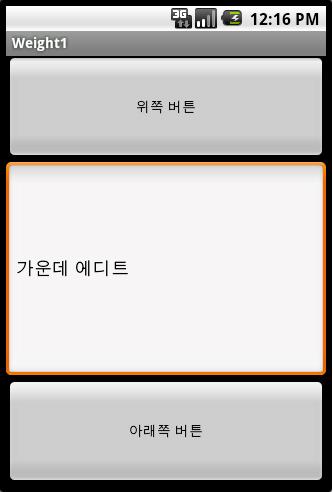
  layout\_weight 속성은 부모 레이아웃의 남은 영역을 얼마나 차지할 거신가를 결정하는 비율 값이다.

  직역하자면 차일드 뷰의 중요도를 의미하는데, 이 값이 높을수록 많은 영역을 차지하게 된다.

  중요도가 0이면 고유의 크기만큼 차지하며, 중요도가 1 이상이면 형제 뷰와의 비율에 따라 부모의 영역을 균등하게 배분한다.

**Layout/weight2.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
      android:orientation="vertical"  
      android:layout\_width="fill\_parent"  
      android:layout\_height="fill\_parent"  
      >  
  <Button  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="위쪽 버튼"  
      android:layout\_weight="1"  
      />  
  <EditText  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="가운데 에디트"  
      android:layout\_weight="3"  
      />  
  <Button  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="아래쪽 버튼"  
      android:layout\_weight="1"  
      />  
  </LinearLayout>



**마진과 패딩**

  레이아웃 안에 뷰가 다닥다닥 붙어있거나, 뷰 안에 내용물이 가득 차 있으면 무척 갑갑해 보일 것이다.

  그러므로 장비의 화면이 아무리 작더라도, 어느 정도 여백을 주어 적당히 띄우는 것이 보기에 좋고 사용하기도 편리하다.

  여백을 주는 방법은 2가지다.

    · padding : 뷰와 내용물의 간격을 지정

    · layouit\_margin : 뷰와 부모와의 간격을 지정

  padding 속성에 값을 대입하면 4면 모두 동일한 여백이 적용되어 한방에 여백을 지정할 수 있다.

  상하좌우 달리 대입하고 싶다면 paddingLeft, paddingTop, paddingRight, paddingBottom 속성에 각각 값을 지정한다.

  ...

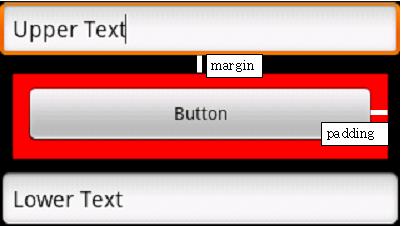
  layout\_margin 속성은 뷰와 부모와의 간격을 지정하면 근처에 형제 뷰도 margin만큼 멀어진다.

  layout\_margin 역시 이 자체에 값을 대입하면 4면 모두 동일한 여백이 적용된다.

  따로 하고 싶다면 layout\_marginLeft, layout\_marginRight, layout\_marginTop, layout\_marginBottom 속성에 대입하면 된다.

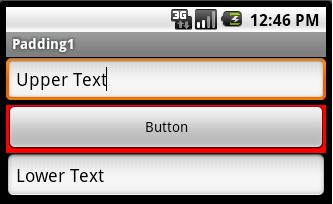
  ...

  패딩과 마진은 둘 다 여백이라는 면에서 유사하지만, 적용되는 위치는 완전히 반대다.



**Layout/padding1.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
      android:orientation="vertical"  
      android:layout\_width="fill\_parent"  
      android:layout\_height="fill\_parent"  
      >  
  **<EditText  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Upper Text"  
      />  
  <LinearLayout  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"  
      android:background="#ff0000"   
      >      
  <Button  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Button"  
      />  
  </LinearLayout>  
  <EditText  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Lower Text"  
      />**  </LinearLayout>



  차일드 뷰끼리 너무 붙어 있어 갑갑하다. 여백을 지정해 넉넉하게 만들자.

**Layout/padding2.xml**

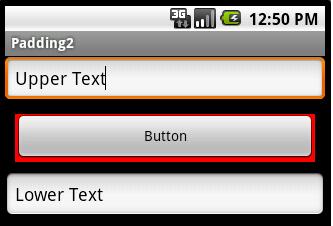
   <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
      android:orientation="vertical"  
      android:layout\_width="fill\_parent"  
      android:layout\_height="fill\_parent"  
      >  
  **<EditText  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Upper Text"  
      />  
  <LinearLayout  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"  
      android:background="#ff0000"**

**android:layout\_margin = "10px"  
      >      
  <Button  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Button"  
      />  
  </LinearLayout>  
  <EditText  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Lower Text"  
      />**  </LinearLayout>

  어떤가? 이제 조금 시원해보이지 않는가?

  그러나, 이것도 뭔가 답답하다.

  모름지기 사내 놈은 속마음이 넓어야 하는데, 이 녀석은 Button이란 놈을 담기에도 벅차보인다.



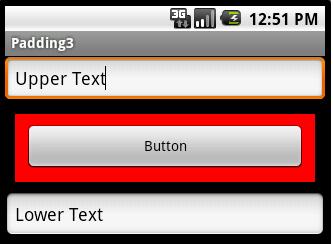
  녀석으 그릇을 넓혀주자.

**Layout/padding3.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
      android:orientation="vertical"  
      android:layout\_width="fill\_parent"  
      android:layout\_height="fill\_parent"  
      >  
  **<EditText  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Upper Text"  
      />  
  <LinearLayout  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"  
      android:background="#ff0000"**

**android:layout\_margin = "10px"**

**android:padding = "10px"  
      >      
  <Button  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Button"  
      />  
  </LinearLayout>  
  <EditText  
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:text="Lower Text"  
      />**  </LinearLayout>



  아, 이제야 속이 다 시원하다.

  지금은 실습 초반이라 일일이 만들어 보고 실행 결과를 캡처하여 보았지만, 이렇게 일일이 하자면 끝이 없을 것이다.

  이후에도 많은 속성을 배울텐데, 이름이 워낙 명시적이라 쉽게 이해할 수 있다.

  간혹 복잡한 속성이 나온다면, 값을 바꿔 비교해가며 차이점을 관찰해보기 바란다.

**<TextVeiw>**

  먼저 strings.xml에 다음과 같은 코딩을 추가한다.

  <?xml version="1.0" encoding="utf-8"?>  
  <resources>  
      <string name="hello">Hello World, AdroidFirst!</string>  
      <string name="app\_name">AdroidFirst</string>  
      **<string name="insa">Hello</string>  
     <string name="anyoung">안녕하세요</string>**  
  </resources>

  위를 보면 문자열의 이름을 insa로 지정하고, 그 내용은 Hello라고 지정했다.

  결국 insa를 부르면 Hello가 출력될 것이다.

  그러나 누가 불러주어야 출력이 될 것이다. 그걸 누가 불러주는가?

  그게 바로 TextView이다.

  main.xml에 추가하도록 하자.

  <?xml version="1.0" encoding="utf-8"?> // xml 문서에서는 주석이 통하지 않으므로 복사 후 제거하고 실행한다.  
<LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    > // 여기선 '>' 이렇게 되어 있는데, 왜 TextView는 '/>' 이렇게 되어 있을까?  
**<TextView    
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:text="@string/insa"  
    android:textColor="#ff0000"  
    android:textSize="20pt"  
    android:textStyle="italic"  
    />  
<TextView    
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:text="@string/anyoung"  
    android:textSize="20sp"  
    android:textColor="#0000ff"  
    />  
<TextView    
    android:layout\_width="fill\_parent"   
    android:layout\_height="wrap\_content"   
    android:text="Good Morning"  
    android:textColor="#8000ff00"  
    android:textSize="5mm"  
    android:typeface="serif"  
    />**</LinearLayout> // 그 숨겨진 뜻은 '>'은 포함한다는 뜻이고, '/>'은 종결됐다는 의미다.

  이제 Ctrl + F11로 실행시키면 다음과 같이 나올 것이다.



**<ImageView>**

  '이미지 뷰' 예제를 실행하기 위해서는 당연히 이미지가 있어야 한다. 예제를 위한 사진은 우측 상단을 확인하라.

  이미지를 옮기는 방법은 2가지가 있다.

  첫째, 자신이 설정한 Workspace로 들어가 drawable 폴더에 직접 넣는 방법이다.

  (1) 자신이 설정한 경로에 Project명을 찾으서 진입하면, 다음과 같은 그림이 나올 것이다.

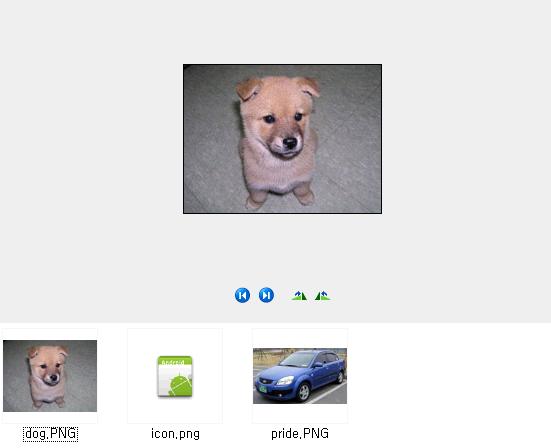
  (1) 여기서 res를 클릭한다.



  (2) hdpi나 lpdi에 넣어도 상관은 없지만, 우리는 mpdi에서 작업하도록 하자.

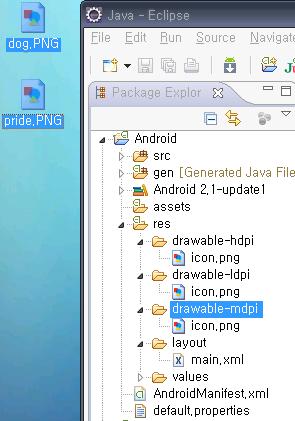


  (3) 이미지 파일을 복사하면 그만이다.



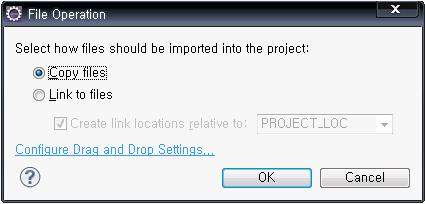
  둘째, 파일을 마우스로 드래그하여 옮기는 방법이다.

  (1) 이미지 파일은 드래그하기 편한 곳에 넣어두고, 클릭한 다음 mdpi로 드래그한다.

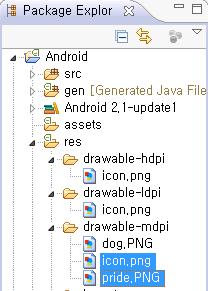


  (2) 드래그하면 물어볼 것이다. '파일을 복사할 것이냐', 아니면 '주소만 연결할 것이냐'라고 묻는다.

       우리는 Copy files를 실행하도록 하자.



  (3) 이제 'OK'를 누르면 복사가 정상적으로 실행된 것을 알 수 있다.



  (이때, 이미지 이름의 첫 글자는 대문자를 쓸 수 없으며, 공백이 들어가서도 안된다. 영문자와 언더바(\_) 정도만 사용할 수 있다)

  이제 그림 파일도 다 옮겼으니, 예제를 수행해보도록 하자.

**Layout/imageviewtext.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
      android:orientation="vertical"  
      android:layout\_width="fill\_parent"  
      android:layout\_height="fill\_parent"  
      >  
  **<ImageView    
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:src="@drawable/pride"  
      />  
  <ImageView    
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:src="@drawable/pride"  
      android:maxHeight="70px"  
      android:maxWidth="120px"  
      android:adjustViewBounds="true"  
      />  
  <ImageView    
      android:layout\_width="fill\_parent"   
      android:layout\_height="wrap\_content"   
      android:src="@drawable/dog"  
      android:tint="#4000ff00"**      />  
  </LinearLayout>

  만약 정상적으로 코드가 이루어졌다면, 이렇게 잘 실행이 되었을 것이다.

  첫번째 이미지는 단순히 src만 지정했으므로, 이미지의 원래 크기대로 출력한다. (src : source의 약자)

  두번째 이미지는 똑같은 src를 지정하되 최대 크기를 120\*70으로 제한했으므로 이 크기에 맞게 축소된다.

  비록 그림이 작아졌지만 종횡비는 유지되므로 자동차가 찌그러져 보이지는 않는다.

  세번째 이미지는 옅은 초록색을 입혀 보았는데, 원본 이미지에 비해 초록색이 살짝 얹힌 효과가 난다.



**<Button&Edit>**

  프로그램이 사용자의 명령을 받아들이려면 버튼, 문자열을 입력받는 에디트 같은 위젯들이 필요하다.

  Button은 사용자가 클릭해 명령을 내릴 수 있는 위젯이며, EditText는 문자열을 입력받는 위젯이다.

**Layout/Buttonedit.xml**

  <?xml version="1.0" encoding="utf-8"?>  
  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"  
      android:orientation="vertical"  
      android:layout\_width="fill\_parent"  
      android:layout\_height="fill\_parent"  
      >  
  <EditText  
      android:layout\_width="wrap\_content"   
      android:layout\_height="wrap\_content"  
      android:id="@+id/edit"   
      android:text="여기다 입력"  
      />  
  <Button  
      android:layout\_width="wrap\_content"   
      android:layout\_height="wrap\_content"   
      android:id="@+id/btn"   
      android:text="입력 완료"  
      />  
  </LinearLayout>

**ButtonEdoit.java**

  package exam.Layout;

  import android.app.\*;   
  import android.os.\*;  
  import android.view.\*;  
  import android.widget.\*;  
  import exam.AndroidExam.\*;

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

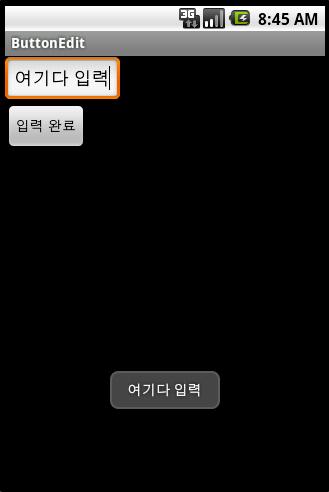
**Button btn = (Button)findViewById(R.id.btn);  
      btn.setOnClickListener(new Button.OnClickListener() {  
        public void onClick(View v) {  
          EditText edit = (EditText)findViewById(R.id.edit);  
          String str = edit.getText().toString();  
          Toast.makeText(ButtonEdit.this, str, Toast.LENGTH\_SHORT).show();  
        }  
      });**    }  
  }

  자바 코딩을 좀 해 본 사람에게는 아주 쉬운 코드이며, 설사 그렇지 않더라도 대략 어떤 내용인지 알 수 있을 것이다.

  버튼의 클릭 이벤트 리스너에서 에디트에 입력된 텍스틑 읽어, 토스트라는 짧은 메세지로 출력한다.

  화면 아래쪽에 임시적인 팝업이 열리고 이 안에 토스트가 나타난다.

  (아직은 이벤트니 리스너니 하는 것을 다룰 단계가 아니니 넘어가도록 하겠다)



  실행을 하고 만들어진 EditText에 입력한 후 입력 완료를 누르면, 아래 사진처럼 입력한 문자가 3초 가량 보이고 사라질 것이다.

////////////////////////////////////////////////////////////////////////////////////////

**<View의 계층>**

  안드로이드 응용프로그램의 화면을 구성하는 주요 단위는 액티비티이다.

  그러나 액티비티 자체는 화면에 보이지 않고, 액티비티 안의 뷰가 사용자를 대면하는 실체이다.

  ...

  View는 안드로이드의 사용자 인터페이스(UI)를 구성하는 핵심 컴포넌트로서 화면상의 사각 영역을 차지하며

  자신의 모양을 그리고 사용자로부터 입력을 받아들인다.

   ...

  View의 파생 클래스 개수는 방대할 뿐 아니라, 각 클래스가 지원하는 속성이나 메서드도 엄청 많아서 상당한 시간이 필요하다.

  그래서 우리는 응용 프로그램을 구성하는 가장 기본적인 것들에 대해 연구해 볼 참이다.

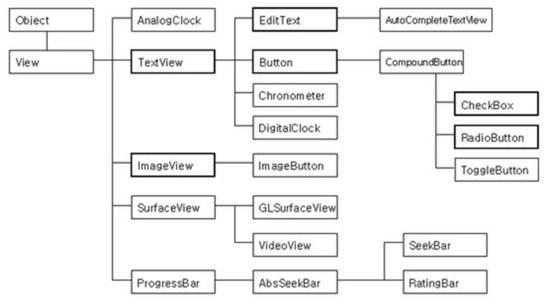
  ...

  View는 크게 다음의 2가지 종류로 구분된다.

   · 위젯 : 직접적으로 보이고, 사용자 인터페이스 구성. 버튼, 라디오, 텍스트 뷰, 에디트, 라디오 버튼 등이 위젯. 흔히 컨트롤이라 불림

   ·  뷰 그룹 : 직접 보이지 않고, 다른 뷰를 담는 컨테이너 역할, 여러 뷰를 유기적으로 모아 놓은 것. 이 부류 클래스를 레이아웃이라 함.

**위젯의 계층**

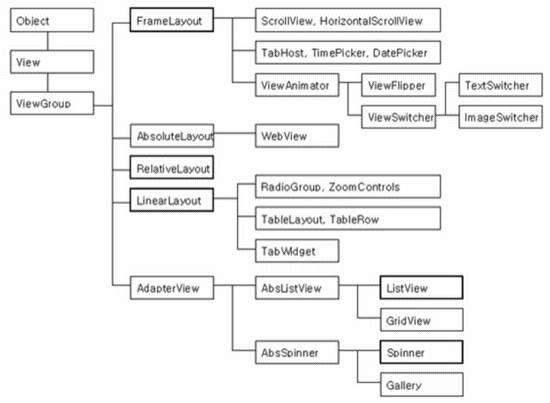


  View도 자바 클래스의 일종이므로 당연히 루트인 Object로부터 파생된다.

  View도 직접적으로 파생되는 모든 클래스가 바로 위젯이며 스스로 그릴 수 있는 능력을 가지고 있다.

  개수가 굉장히 많은데 굵게 표시해 놓은 클래스가 빈번히 사용되는 것들이므로 이 클래스들을 우선적으로 학습하도록 하자.

**뷰의 계층**



  뷰 그룹은 View로부터 파생된 ViewGroup의 서브 클래스로서 다른 뷰들을 차일드로 포함하며 정렬하는 기능을 가진다.

  편의상 위젯과 뷰 그룹으로 나누어 보았는데, 어떤 클래스는 뷰 그룹이면서도 위젯처럼 사용되기도 한다.

  물론 단순 위젯에 비해 여러 개의 뷰가 모인 것이므로 아무래도 기능이 많고 좀 더 복잡하다.

  대표적으로 항목의 집합을 표시하는 리스트 뷰는 다른 뷰를 포함하는 뷰 그룹이면서 사용자를 직접적으로 대면한다.

  ...

  특정 위젯을 상세하게 연구하고 싶다면 그 슈퍼 클래스들부터 연구해야 한다.

  알다시피 서브 클래스는 슈퍼 클래스의 모든 속성을 상속받으므로 슈퍼 클래스의 속성들을 미리 파악하면 서브클래스가 쉬워진다.

  예를 들어 Button을 자유자재로 사용하고 싶다면 상위 클래스인 TextView와 View의 모든 멤버들에 대해 파악해야 한다.

  상위 클래스를 정복했으면 형제 클래스도 비슷하므로 거저먹기다. 그래서 클래스 계층을 잘 알아 두는 것이 중요하다.

**<View의 속성>**

  모든 뷰는 View 클래스로부터 파생되므로 View가 가진 모든 속성과 메서드를 그대로 상속받는다.

  따라서 View에 정의된 멤버는 파생 클래스에도 공통적으로 존재한다.

  최상위 뷰에 정의된 멤버들은 사용 빈도가 아주 높으며 또한 그만큼 중요한 의미를 가진다.

  ...

  View 클래스의 정의된 속성만 해도 수십 개가 넘고, 처음부터 모든 속성을 일일이 공부하기는 어렵다.

  이 속성들에 대한 상세한 설명은 레퍼런스의 View 클래스에서 볼 수 있다.

  다행히 속성의 이름이 아주 명시적이어서 이름만으로 의미를 파악하기 쉽게 되어 잇다.

  속성명은 XML 문서에도 그대로 적용된다.

  ...

**id란?**

  뷰를 칭하는 이름이다. R.java에서 모든 주소를 관리하는데, 등록은 사용자가 직접 해주어야 한다.

  등록 방법은 android:id="@+id/name" 이렇게 한다.

  정의는 이렇다.

    @ : 참조한다는 뜻이므로, 반드시 붙여줘야 한다.

    + : ID를 새로 정의한다. 등록한다는 뜻이다. 그러므로 처음 정의 때만 붙이고, 참조할 때는 붙이지 않는다.

    id : 예약어이니 그대로 쓰면 되고, /name은 name이라는 아이디로 등록하는 것이다. 이름은 마음대로 붙이면 된다.

  XML 문서에 ID를 지정해 놓으면 이 이름이 R.java에 정수형 상수로 정의된다.

  코드에서 이 뷰를 참조할 때는 findViewById라는 메서드를 통해 호출한다.

  ...

  id는 뷰의 이름을 정의하는 중요한 속성이긴 하지만 모든 뷰에 id를 의무적으로 지정할 필요는 없다.

  예를 들어 배경에 이미지를 출력하는 이미지 뷰는 XML 문서에 배치되어 있기만 하면 될 뿐 이후에 참조할 일이 없으므로

  굳이 id를 줄 필요가 없다.

  ...

**layout\_width, layout\_height**

  뷰의 폭과 높이를 지정한다. 뷰는 부모 뷰(또는 액티비티) 안에 배치되는데 이 속성이 지정하는 만큼의 크기로 배치된다.

  수평, 수직 각 방향에 대해 크기를 지정할 수 있으며 다음 세 가지 중 하나의 값을 가진다.

|  |  |
| --- | --- |
| **속성값** | **설명<?xml:namespace prefix = o />** |
| fill\_parent | 부모의 주어진 크기를 다 채운다. |
| wrap\_content | 내용물의 크기만큼만 채운다. |
| 정수 크기 | 지정한 크기에 맞춘다. |

  보통 가득 채우거나. 아니면 자기 폭에 맞추거나 둘 중의 하나이다.

  물론 지정한 크기가 액면대로 다 받아지는 것은 아니고, 주변에 다른 위젯이 있다면 폭을 나누어 쓰이기도 하고, 덮이기도 할 것이다.

  특수한 경우 명시적인 크기를 지정할 수 있는데, 이때는 정수 하나와 단위를 지정하는 예약어를 같이 사용한다.

  이 단위들은 크기를 지정하는 모든 속성에 공통적으로 적용된다.

|  |  |
| --- | --- |
| **단위** | **설명** |
| px | 픽셀 |
| in | 인치 |
| mm | 밀리미터 |
| pt | 포인트 |
| dp(또는 dip) | 해상도에 독립적인 단위 |
| sp(또는 sip) | 폰트 가변 크기 |

  이 단위들은 크기를 지정하는 모든 속성에 공통적으로 적용된다.

  물리적인 픽셀 단위로 지정할 수도 있고, 인치나 밀리미터 같은 논리적 단위로도 지정할 수 있다.

  px는 가장 직관적이나, 해상도에 따라 크기가 달라지므로 가급적 논리 단위를 사용하는 것이 좋다.

  dp 단위로 지정하면 해상도만큼 길이가 늘어나 해상도에 상관없이 비슷한 크기로 보인다.

**background**

  뷰의 배경을 어떻게 채울 것인가를 지정한다.

  단순히 색상 뿐 아니라, 버튼이나 입력 가능한 사각 박스로도 지정할 수 있다.

  색상은 #ff0000 (빨간색), #0000ff (파란색) 등 이처럼 16진수로 정의하고, 모든 속성에 이 형식이 적용된다.

   #RGB

   #ARGB

   #RRGGBB

   #AARRGGBB

  '#' 다음에 16진수로 각 색상 요소의 강도를 지정한다. 웹에서 흔히 사용되는 표준화된 방식이다.

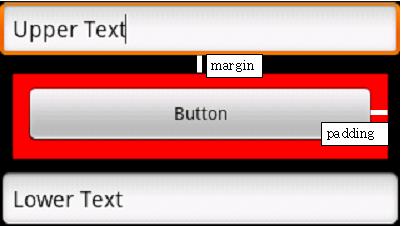
  각 요소당 한 자리만 쓸 수도 있지만, 통일성을 위해 가급적 두 자리로 쓰는 것이 안전하다.

**padding**

  뷰와 내용물 간의 간격을 뜻한다.

  padding 속성에 값을 지정하면 4방향에 대해 동일한 여백이 적용되며

  padding Left, padding Top, padding Right, padding Bottom 속성에 개별적으로 값을 지정해 서로 다른 여백을 줄 수 있다.



  margin은 일단 모른 척 하라.

**Visibility**

  뷰의 유무 표시이다.

  디폴트 값은 당연히 보이지만, 속성을 지정하면 얘기가 달라진다.

|  |  |
| --- | --- |
| **속성값** | **설명** |
| visible | 보이는 상태이다. |
| invisible | 숨겨진 상태이되 자리는 차지한다. |
| gone | 숨겨지며 자리도 차지하지 않는다. |

  보이기 상태는 런타임에도 얼마든지 변경할 수 있다.

**clickable, longClickable**

  마우스 클릭 이벤트를 받을 것인지를 지정한다.

  한마디로 클릭하느냐, 오래 클릭하느냐. 이것을 입력으로 받아줄 것인지 결정한다.

  진위형이므로 true 또는 false 둘 중 하나의 값을 지정한다.

**focusable**

  키보드 포커서를 받을 수 있는지 지정한다.

  뷰의 디폴트 값은 false이므로, 키 입력을 받으려면 true로 변경해야 한다.

  에디트나 버튼처럼 사용자의 입력을 받아야 하는 파생 클래스는 디폴트가 true로 지정되어 있다.

  ...

  이 속성들에 대한 예제는 잠시 후에 만나보도록 하자.

**<TextView>**

  TextView는 이름 그대로 화면에 텍스트를 출력하는 위젯이다.

  단순히 문자열을 보여줄 뿐, 입력을 받아들이지 않는다.

  고정된 텍스트나 다른 위젯의 제목을 보여줄 때 사용한다.

  단순해 보여도 굉장히 많은 속성을 제공한다.

**text**

  텍스트 뷰의 가장 중요한 속성으로 출력할 문자열 지정한다.

|  |  |
| --- | --- |
| **형식** | **설명** |
| "문자열" | 겹 따옴표로 문자열을 싸서 바로 대입한다. \ 문자가 들어가면 이스케이프된다. \n은 개행이며 \uxxxxx는 유니코드 문자이다. |
| @[패키지:]type:name | 리소스에 대한 레퍼런스로 지정한다. 보통 string.xml에 문자열을정의해 놓고 @string/id 식으로 지정한다. |
| ?[패키지:][type:]name | 테마 속성으로 지정한다. |

  text="안녕하세요" 식으로 리터럴 문자열을 바로 지정하는 것이 쉽고 직관적이나,

  다국어 버전을 만들 경우에는 직접 지정하기보다, strings.xml 문자열을 정의해놓고 id를 지정하는 것이 유리하다.

**textColor**

  문자열의 색상을 지정한다. #rrggbb나 #aarrggbb 형식으로 각 색상 요소들의 강도를 지정한다.

  디폴트 색상은 불투명한 밝은 회색이다.

**textSize**

  텍스트의 폰트 크기를 지정한다.

  실수 타입으로 정밀한 크기를 지정할 수 있으며 숫자 뒤에 sp, dp, px, in, mm 등의 단위를 같이 지정해야 한다

  텍스트는 폰트 크기에 따라 가변적인 sp 단위를 쓰는 것이 가장 합리적이다

  (예) 16px, 10pt, 15sp

**textStyle**

  폰트의 속성을 지정한다. normal, bold, italic 중 하나를 쓰거나 '|'로 묶어 두 개 이상을 사용할 수도 있다.

  이때, 앞뒤로 공백이 있어서는 절대 안된다.

**typeFace**

  글꼴의 모양을 지정한다. 모바일은 폰트 개수에 제약이 있으므로 noramal, sans, serif, monospace 중 하나로 선택할 수 있다.

**width, height**

  텍스트의 뷰의 폭과 높이이며, 크기 값과 단위를 같이 지정한다.

  하지만 거의 사용되지 않는다. 절대적인 크기보다는 상대적인 크기가 범용성도 높고, 더 중요하다.

**singlieLine**

  텍스트가 위젯의 폭보다 더 길 때 강제로 한 줄에 출력하도록 한다.

  이 속성의 디폴트는 false로 폭보다 더 긴 줄은 자동으로 아래쪽으로 개행된다.

  고로 꼭 한 줄에 출력하고 싶다면 true로 변경한다.

  ...

**<Image View>**

  이미지 뷰는 아이콘이나 비트맵을 출력하는 위젯이다.

  다양한 경로의 이미지를 공급받을 수 있다. 리소스, 파일 등은 물론, 컨텐트 프로바이더나 웹상의 이미지도 표시할 수 있다.

  크기나 색조를 지정할 수 있는 다음과 같은 속성들이 제공된다.

**src**

  출력할 이미지를 지정하는 가장 중요한 속성이다.

  ##rrggbb 형태의 색상 값을 출력할 수도 있지만, 가장 흔한 경우는 리소스에 준비한 이미지를 출력하는 것이다.

  리소스에 이미지를 복사해 두고, @drawable/ID 형식으로 이미지의 ID를 지정하면 해당 이미지가 이미지 뷰의 표면에 출력된다.

**maxHeight, maxWidth**

  이미지가 출력될 최대 크기를 지정한다.

  모바일의 해상도가 충분치 않으므로, 최대 크기를 적당히 제한할 필요가 있다.

  크기가 강제적으로 적용되는 것은 아니며, 지정한 크기에 최대한 맞추되 이미지의 종횡비는 가급적 유지하도록 되어 있다.

**adjustViewBonus**

  이미지의 종횡비를 맞추기 위해 이미지 뷰의 크기를 적당히 조정한다.

  true나 false, 둘 중의 하나를 선택해야 한다.

  그러므로 maxHeight, maxWidth와 실과 바늘처럼 같이 따라다닌다.

**cropToPadding**

  true일 경우 위젝의 주어진 여백에 맞추기 위해 이미지의 일부를 잘라낸다.

  이 속성을 사용할 경우, 이미지 전체가 보이지 않을 수도 있다.

**tint**

  반투명이다.

  #aarrggbb 형식으로 색상을 지정하는데, 이 색상이 이미지 위에 살짝 덮혀 출력된다.

**scaleType**

  (이미지의 원래 크기와 다르게 출력할 때) 적용할 확대, 축소 알고리즘을 지정한다.

  matrix, fitXY, center, centerCrop, centerInside 등의 여러가지 알고리즘 중 하나를 지정한다.

  각 알고리즘의 정의는 그래픽 관련 서적을 봐야 할 정도로 전문적인 것이라 여기서 차이점을 설명하기는 어렵다.

  이 속성들 중 src는 단연 가장 압도적인 사용을 보인다. 나머지는 드물게 사용된다.

  이미지를 출력하려면, 프로젝트의 폴더 drawable 폴더로 파일을 복사해두어야 한다.

  그 후 이 파일을 레이아웃이나 코드에서 사용할 수 있다.

  안드로이드에서 공식적으로 지원하는 이미지 포맷은 jpg, png, gif 등이다.

   · gif : 품질이 좋지 않고, 라이센스 문제까지 있으니 권장하지 않는다.

   · jpg : 일반적인 그래픽 포맷이지만 그보다는 png를 권한다.

   · png : 알파 채널이 있어 반투명을 지원하며 직사각형이 아닌 이미지도 만들 수 있어 실용성이 높다.

  SDK 1.5 이전 버전에는 drawable 폴더 하나만 있었으나, 이제 해상도별로 제공할 수 있도록 세 개가 제공이 된다.



  hdpi : 240 정도의 고해상도를 의미

  ldpi : 160 정도의 고해상도를 의미

  mdpi : 120 정도의 고해상도를 의미

  각 폴더에 이미지를 넣어두면, 장비의 크기와 해상도에 따라 운영체제가 알아서 결정한다.

**<버튼과 에디트>**

  텍스트 뷰와 이미지 뷰는 화면에 문자열이나 그림을 단순히 보여주기만 하는 정적이니 위젯이며 사용자와의 상호 작용은 없다.

  프로그램이 사용자의 명령을 받아들이려면 명령을 전달받는 버틑, 문자열을 입력받는 에디트 같은 위젯들이 필요하다.

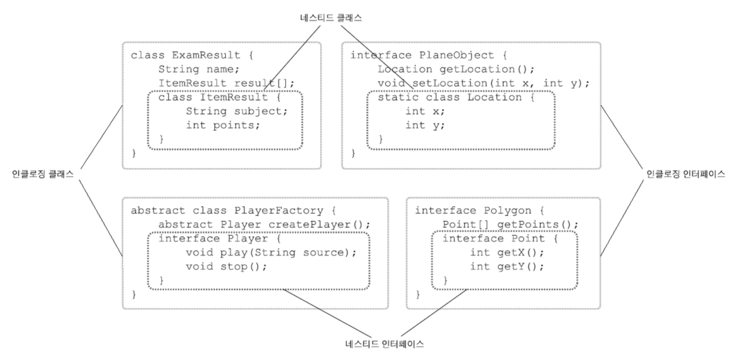
이에 대한 내용은 다음 게시물의 예제를 통해 알아보도록 하겠다.

//////////////////////////////////////////////////////////////////////////////

지금까지 작성했던 클래스와 인터페이스는 모두 다른 클래스나 인터페이스에 종속이지 않은 독립적인 클래스와 인터페이스만 다루었다.

자바에서는 클래스안이나 인터페이스안에 클래스를 선언하거나 인터페이스를 선언하는 문법도 가능하다.

아래 그림처럼 네스티드 클래스를 포함하고 있는 클래스를 인클로징클래스 또는 outer 클래스라고 한다.



안드로이드 개발 시 만들어지는 R 클래스내에 id, drawable, layout, string등 nested 클래스에 대한 정의가 만들어지는데 이러한 클래스들을 static nested class라고 부른다.

|  |
| --- |
| package sample;    public final class R {       public static final class drawable {             public static final int icon=0x7f020000;       }       public static final class id {             public static final int sclView=0x7f050000;       }       public static final class layout {             public static final int main=0x7f030000;       }       public static final class string {            public static final int app\_name=0x7f040001;            public static final int hello=0x7f040000;       } } |

Nested 클래스를 사용하는 목적은 다음과 같다.

- 오직 하나의 클래스에서만 사용되어질 경우 사용하는 클래스 내에 사용되는 클래스의 정의를 포함시켜 논리적으로 그룹핑하는 것이다.

- 캡슐화를 증대시키는 목적으로도 사용한다. 즉, A라는 클래스에 private으로 선언된 멤버(변수와 메소드)를 B라는 클래스에서 사용해야 할 경우 B 클래스를 A클래스 내에 정의한다. 만약 B 클래스가 static nested 클래스일 경우에는 A 클래스의 전역 멤버(static 선언)만 접근할 수 있다.

|  |
| --- |
| public class OuterClass {        private String name;        public static class InnerClass{            public void printName(){                    System.out.println(name); // 컴파일 에러            }        } } |

- 클래스 내에 Nested 클래스를 사용하는 코드 영역과 가까이 있게 하여 소스의 가독성과 유지보수성을 높이는데 목적이 있다.

**Nested 클래스의 인스턴스를 생성할 때 static nested 클래스와 inner 클래스가 방식이 다르다.**

**static nested 클래스는 new [Outer 클래스 명].[nested 클래스 명]()으로 바로 생성하지만**

**inner 클래스는 먼저 Outer 클래스에대한 인스턴스를 생성 한 후 outerInstance.new [inner 클래스명]()으로 생성해야 한다.**

|  |
| --- |
| public class OuterClass {     private String name;     public void setName(String name){         this.name = name;     }     public class InnerClass{         public void printName(){               System.out.println(name); // 컴파일 에러        }     }     public static class StaticNestedClass{         public void printName(){              System.out.println("static nested class");         }     } }      public class NestedMain {      public static void main(String[] args){             OuterClass outer = new OuterClass();             outer.setName("Outer class");              // Inner 클래스는 outer 클래스의 인스턴스를 생성한 후 생성 가능            OuterClass.InnerClass inner = outer.new InnerClass();            inner.printName();              // static nested 클래스는 바로 생성 가능             OuterClass.StaticNestedClass staticClass =   new OuterClass.StaticNestedClass();             staticClass.printName();      } }    <실행> Outer class static nested class |

**1. 네스티드 클래스의 선언과 이용**

네스티드 클래스는 선언위치와 선업방법에 따라 세가지 종류로 나뉜다.

* 정적 네스티드 클래스(static nested class)
* 이너 클래스(Inner class)
* 로컬 이너 클래스(local inner class)

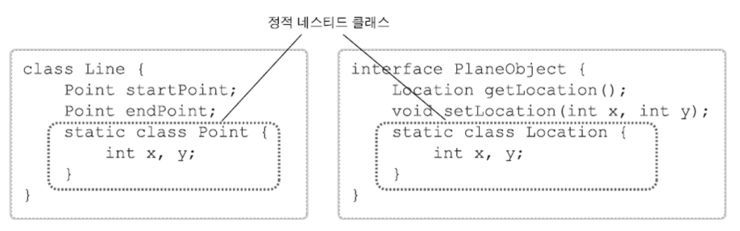
**1-1. 이너 클래스**

예제] 인터넷 쇼핑몰에서 사용하는 상품명, 수량, 단가, 금액을 출력하는 장바구니 데이터를 클래스로 표현한다고 가정하고, 이 데이터들 중에서 상품명, 수량, 단가, 금액은 반복되는 항목들이다.

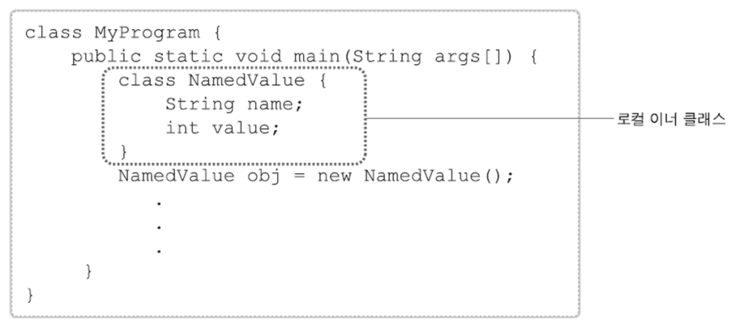
이처럼 반복되는 데이터는 하나로 묶어서 장바구니 클래스 안에 이너클래스로 선언할 수 있따.

|  |
| --- |
| public class Cart {      ArrayList<Item> list = new ArrayList<Item>();      class Item{      //이너클래스            String name;            int num;            int price;            Item(String name, int num, int price) {                 this.name = name;                 this.num = num;                 this.price = price;            }               int getTotal(){                 return num \* price;           }       }         //Item 이너클래스에 값 초기화       void addItem(String name, int num, int price){            Item item = new Item(name, num, price);            list.add(item);              //list.add(new Item(name,num, price));       }         void addItem(Item item){            list.add(item);      }        Item getItem(int index){           return list.get(index);      } } |

**1-2. 정적 네스티드 클래스**



**1-3. 로컬 이너 클래스(local inner class)**



**1-4. Anonymous 클래스**

Anonymous 클래스는 코드상에서 오직 한번만 인스턴스로 생성하기 위해 정의하는 클래스로서 보통 이벤트의 리스너 인스턴스를 생성할 때 사용한다. 익명클래스는 new 수식의 연장이므로 끝에 꼭 세미콜론(;)을 붙여주어야 한다.

new 클래스명() {

클래스 정의

};

또는

new 인터페이스명() {

인터페이스 정의 및 구현

};

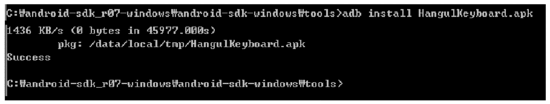
**/////////////////////////////////////////////////////////////////////////////////////////////////**

**안드로이드 한글 키보드 설정**

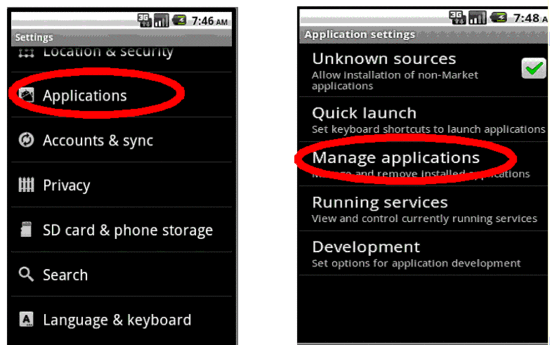
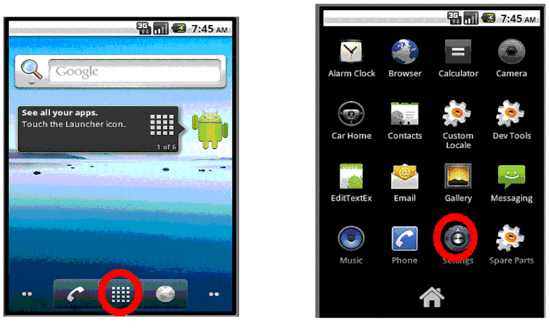
**1. 제공되는 HangulKeyboard.apk파일을 내려받아 안드로이드 SDK의 tools경로(C:\android-sdk\_r07-windows\android-sdk-windows\tools)에 복사한다.**

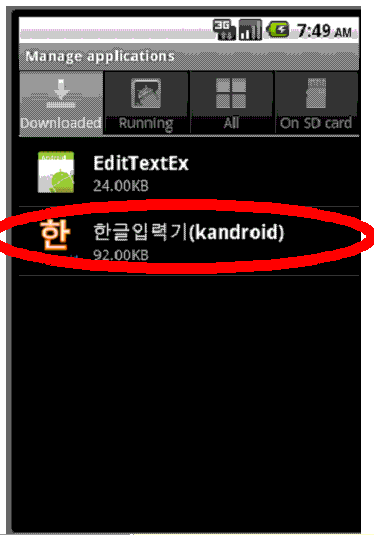
**2. cmd 창에서 다음과 같이 실행 시킨다.**

**(adb install HangulKeyboard.apk)**



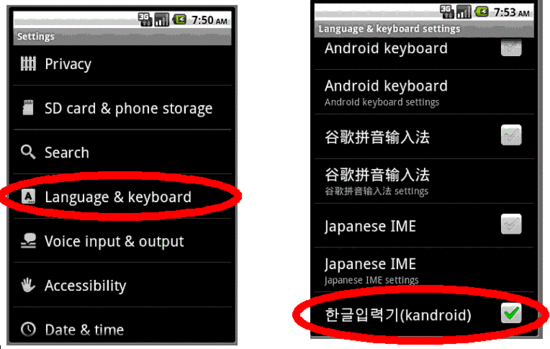
**3. 에뮬레이트에 한글입력기가 설치 된것을 확인한다.**





**4. Language&Keyboard 메뉴에서 한글 키보드로 설정한다.**

**(기존에 설정 되어 있는 것을 모두 해제 한 후 한글입력기를 체크하도록 한다. )**



**5. Custom Locale에서 한국어로 설정한다.**

