

5. 이벤트 처리

1. 안드로이드에서 이벤트 처리 방법

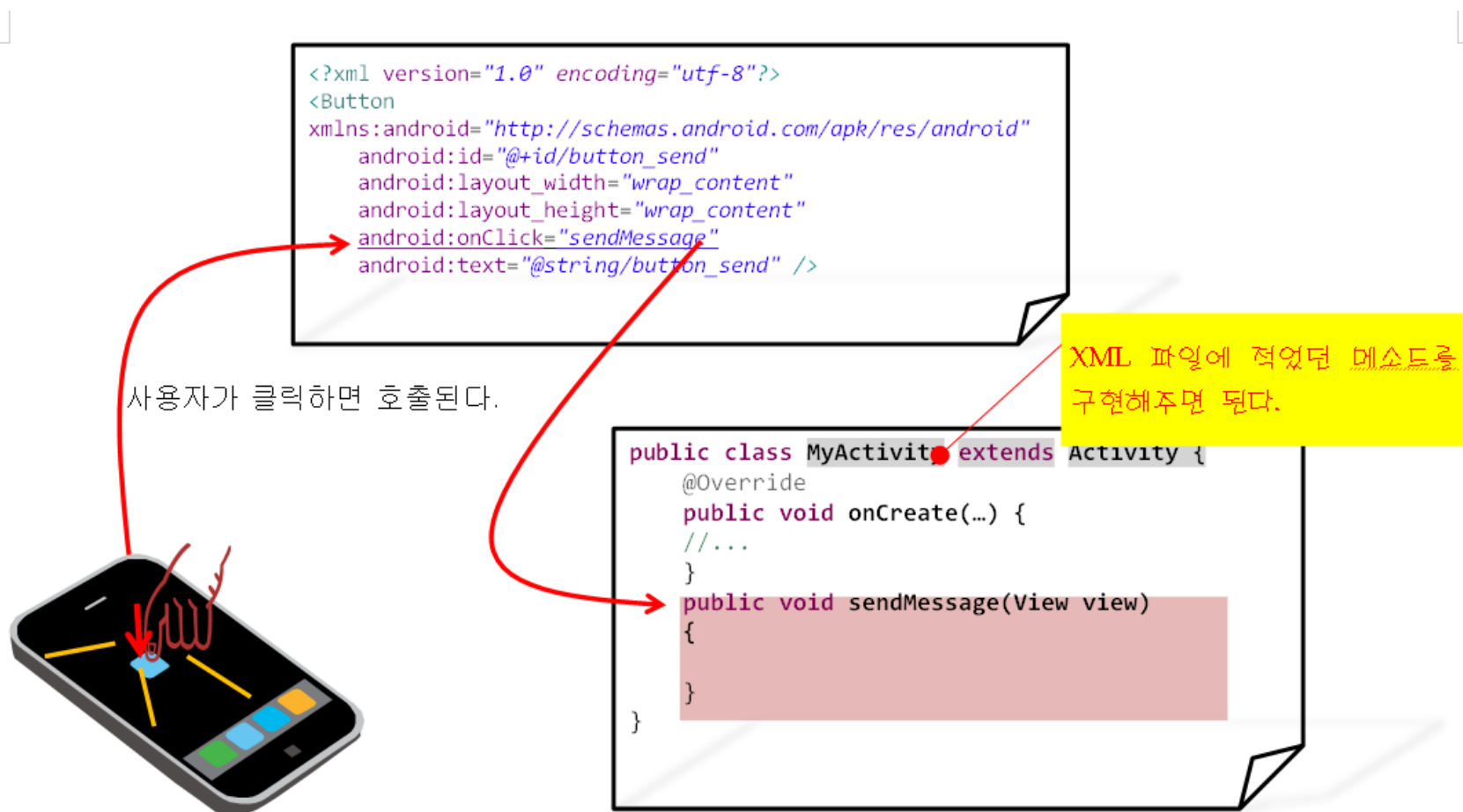
목차

- 안드로이드에서 이벤트 처리 방법
- XML로 이벤트처리
- 이벤트 처리 메소드 재정의
- 이벤트 처리 객체 사용
- 키 이벤트
- 터치 이벤트
- 멀티터치 이벤트

1. 안드로이드에서 이벤트 처리 방법

□ XML을 이용한 이벤트 처리

▣ XML 리소스에 이벤트 함수 등록



<ImageButton> 태그를 사용한 이미지 버튼

main.xml

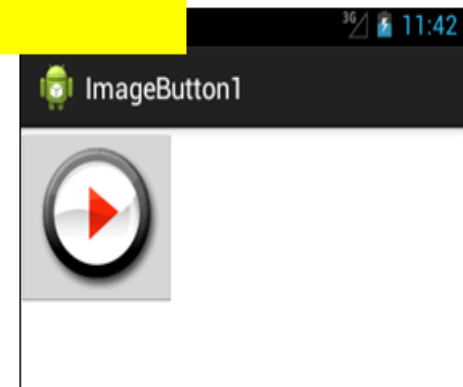
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
```

```
    <ImageButton
        android:id="@+id/imageButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="onClick"
        android:src="@drawable/mybutton" />
```

```
</LinearLayout>
```

이미지 버튼을 정의한다.

XML로 이벤트 처리 메소드를 지정하
자. 이것이 가장 간단하다.



이벤트 처리 코드

ImageButtonActivity.java

```
package kr.co.company.imagebutton1;  
// 소스만 입력하고 Ctrl-Shift-O를 눌러서 import 문장을 자동으로 생성한다.
```

```
public class ImageButtonActivity extends Activity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }
```

```
    public void onClick(View target) {  
        Toast.makeText(getApplicationContext(), "버튼이 눌러졌습니다",  
            Toast.LENGTH_SHORT).show();  
    }
```

이미지 버튼이 눌려지면 토스트 메시지를 출력한다.

```
}
```

이벤트 처리 객체 사용



사용자가 클릭하면 호출된다.

```
Class MyClass
{
    class Listener implements OnClickListener {
        public void onClick(View v) {
            ...
        }
    }
    ...
    Listener lis = new Listener();
    button.setOnClickListener(lis);
    ...
}
```

버튼에 붙은 리스너 객체가 이벤트를 처리한다.

이벤트 리스너



전체적인 구조

```
class MyClass
```

```
{
```

```
    class Listener implements OnClickListener {
```

```
        public void onClick(View v){
```

```
            ...
```

```
        }
```

```
    }
```

```
    ...
```

```
    Listener lis = new Listener();
```

```
    button.setOnClickListener(lis);
```

```
    ...
```

```
}
```

인터페이스를 구현한 클래스 정의

이벤트 리스너 객체 생성

버튼에 이벤트 리스너 객체를 등록

리스너의 종류

리스너	콜백 메소드	설명
<u>View.OnClickListener</u>	<u>onClick()</u>	사용자가 어떤 항목을 터치하거나 네비게이션 키나 트랙볼로 항목으로 이동한 후에 엔터키를 눌러서 선택하면 호출된다.
<u>View.OnLongClickListener</u>	<u>onLongClick()</u>	사용자가 항목을 터치하여서 일정 시간 동안 그대로 누르고 있으면 발생한다.
<u>View.OnFocusChangeListener</u>	<u>onFocusChange()</u>	사용자가 하나의 항목에서 다른 항목으로 포커스를 이동할 때 호출된다.
<u>View.OnKeyListener</u>	<u>onKey()</u>	포커스를 가지고 있는 항목 위에서 키를 눌렀다가 놓았을 때 호출된다.
<u>View.OnTouchListener</u>	<u>onTouch()</u>	사용자가 터치 이벤트로 간주되는 동작을 한 경우에 호출된다.
<u>View.OnCreateContextMenuListener</u>	<u>onCreateContextMenu()</u>	컨텍스트 메뉴가 구축되어 있는 경우에 호출된다.

리스너 객체를 생성하는 방법

- 리스너 클래스를 내부 클래스로 정의.
- **리스너 클래스를 무명 클래스로 정의.**
- 리스너 인터페이스를 액티비티 클래스에 구현.



가장 많이 사용되는 방법!

무명 클래스

- 클래스 몸체는 정의되지만 이름이 없는 클래스이다.
- 무명 클래스는 클래스를 정의하면서 동시에 객체를 생성하게 된다.

```
class ClickListener implements OnClickListener {  
    ...  
}  
obj = new ClickListener();
```



```
obj = new OnClickListener() { .... };
```

코드 분석

...

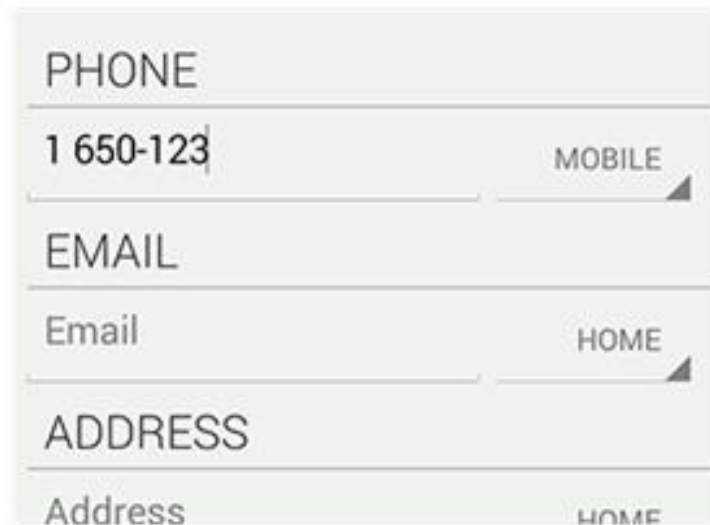
```
public class ButtonEvent2Activity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        Button button = (Button) findViewById(R.id.button);  
        button.setOnClickListener(new OnClickListener() {  
            public void onClick(View v) {  
                Toast.makeText(getApplicationContext(), "버튼  
                    눌러졌습니다", Toast.LENGTH_SHORT).show();  
            }  
        });  
    }  
}
```

무명 클래스 사용



텍스트 필드

- 텍스트 필드(text field)를 사용하면 사용자가 앱에 텍스트를 타이핑하여 입력할 수 있다.
- 단일 라인이거나 멀티 라인일 수 있다.



PHONE

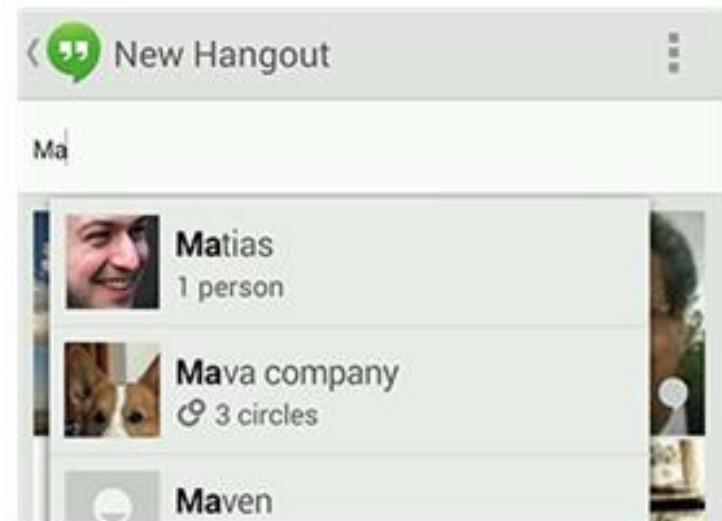
1 650-123 MOBILE

EMAIL

Email HOME

ADDRESS

Address HOME



키보드 종류 지정

`<EditText`

```
android:id="@+id/email_address"  
android:layout_width="fill_parent"  
android:layout_height="wrap_content"  
android:hint="@string/email_hint"  
android:inputType="textEmailAddress" />
```

이메일 형태의 입력을 받는다.



(`textEmailAddress` 입력 타입)



(`phone` 입력 타입)

에디트 텍스트

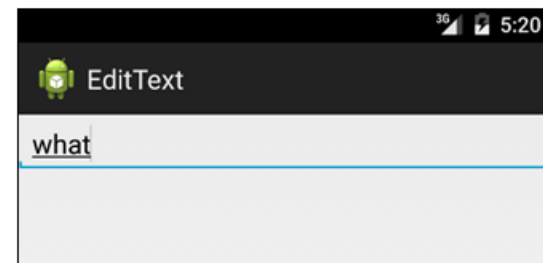
main.xml

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

```
    <EditText
        android:id="@+id/search"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Search"
        android:imeOptions="actionSend"
        android:inputType="text" />
```

에디트 텍스트

```
</LinearLayout>
```



에디트 텍스트의 이벤트 처리

EditTextActivity.java

```
package kr.co.company.editttext;
```

// 소스만 입력하고 Alt+Enter를 눌러서 import 문장을 자동으로 생성한다.

```
public class EditTextActivity extends ActionBarActivity {
```

```
    @Override
```

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

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

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

```
        final EditText editText = (EditText) findViewById(R.id.search);
```

```
        editText.setOnEditorActionListener(new TextView.OnEditorActionListener() {
```

```
            @Override
```

```
            public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
```

```
                boolean handled = false;
```

```
                if (actionId == EditorInfo.IME_ACTION_SEND) {
```

```
                    Toast.makeText(getApplicationContext(), editText.getText(),
```

```
                        Toast.LENGTH_SHORT).show();
```

```
                    handled = true;
```

```
                }
```

```
                return handled;
```

```
            }
```

```
        });
```

```
    }
```

```
}
```

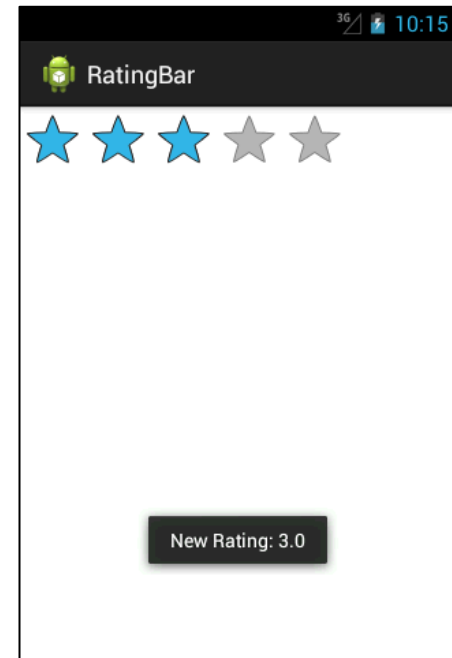
사용자가 "Send" 버튼을 누르면 화면에 토스트 메시지를 표시한다.

여기에 final이 붙는 이유는 참고사항에 있음!

레이팅바

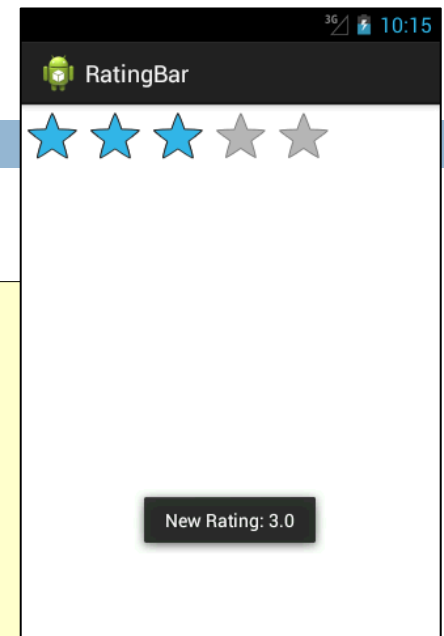
□ XML로 레이팅바 정의한다.

```
...  
<RatingBar  
    android:id="@+id/ratingbar"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:numStars="5"  
    android:stepSize="1.0"  
</>
```



레이팅바

```
...
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        final RatingBar ratingbar = (RatingBar) findViewById(R.id.ratingbar);
        ratingbar.setOnRatingBarChangeListener(new OnRatingBarChangeListener() {
            public void onRatingChanged(RatingBar ratingBar, float rating,
                boolean fromUser) {
                Toast.makeText(getApplicationContext(),
                    "New Rating: " + rating, Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```



커스텀 컴포넌트

- 개발자가 직접 View 클래스를 상속받아서 필요한 위젯을 개발



이벤트 처리 메소드 재정의

사용자가 키를 누르면 호출된다.



```
class MyView extends View
{
    ...
    boolean onKeyDown()
    {
        ...
    }
    ...
}
```

재정의할 수 있는 콜백 메소드

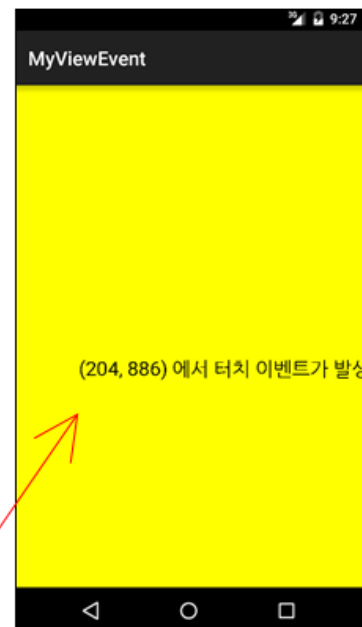
- ❑ **onKeyDown(int, KeyEvent)**
- ❑ **onKeyUp(int, KeyEvent)**
- ❑ **onTrackballEvent(MotionEvent)**
- ❑ **onTouchEvent(MotionEvent)**
- ❑ **onFocusChanged(boolean, int, Rect)**

예제

```
class MyView extends View {  
    int key;  
    String str;  
    int x, y;  
  
    public MyView(Context context) {  
  
        super(context);  
        setBackgroundColor(Color.YELLOW);  
    }  
}
```

```
@Override  
public boolean onTouchEvent(MotionEvent event) {  
    x = (int) event.getX();  
    y = (int) event.getY();  
    invalidate();  
    return super.onTouchEvent(event);  
}
```

```
@Override  
protected void onDraw(Canvas canvas) {  
    Paint paint = new Paint();  
    paint.setTextSize(60);  
    canvas.drawText("(" + x + " ", "+y+") 에서 터치 이벤트가 발생하였음", x, y, paint);  
}
```

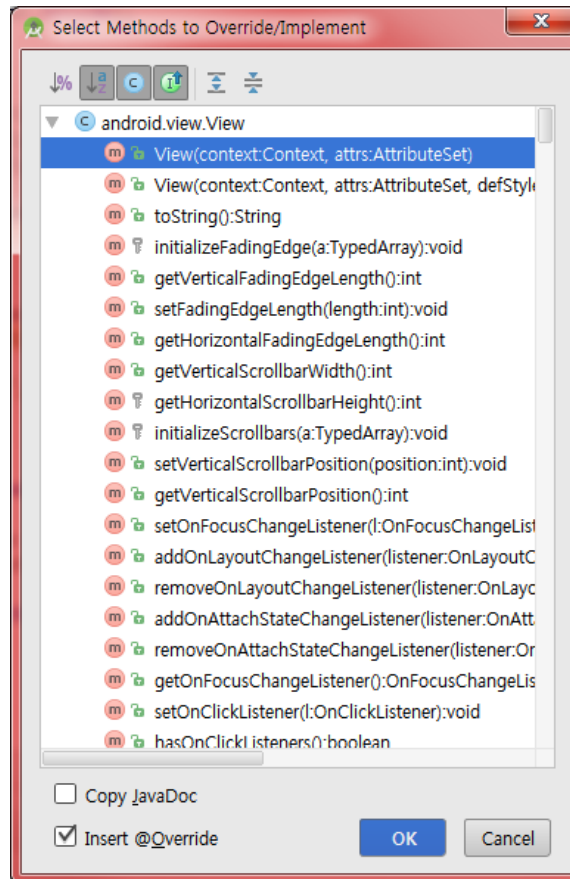


예제

```
public class MyViewEventActivity extends ActionBarActivity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        MyView w = new MyView(this);  
        setContentView(w);  
    }  
}
```

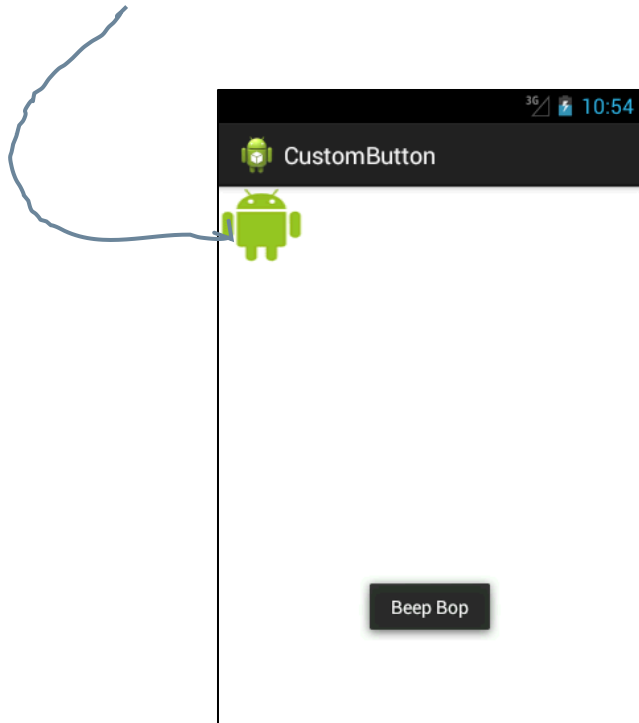
콜백 메소드를 재정의할 때 편리한 기능

- [Code]->[Override Methods...] 사용!



커스텀 버튼

- 버튼 위에 텍스트 대신에 이미지가 그려져 있는 버튼

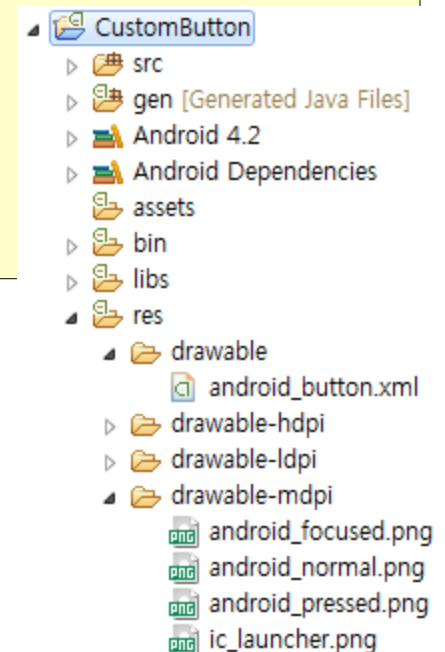


커스텀 버튼 정의

- XML로 버튼에 사용되는 이미지를 등록한다.

/res/drawable/android_button.xml

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/android_pressed"
        android:state_pressed="true" />
  <item android:drawable="@drawable/android_focused"
        android:state_focused="true" />
  <item android:drawable="@drawable/android_normal" />
</selector>
```



레이아웃 파일

- 레이아웃 파일에 버튼을 정의한다.

/res/layout/main.xml

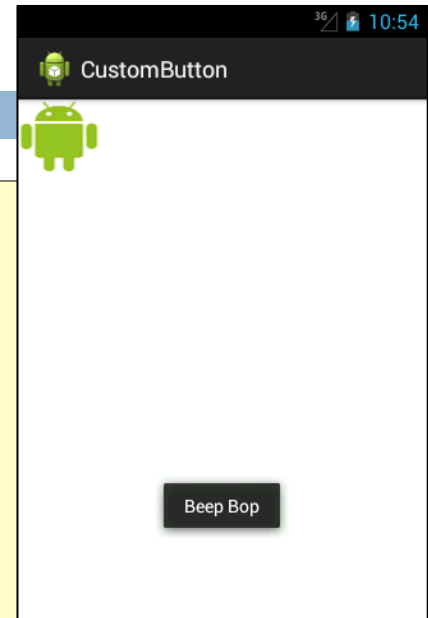
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/android_button"
        android:padding="10dp" />

</LinearLayout>
```

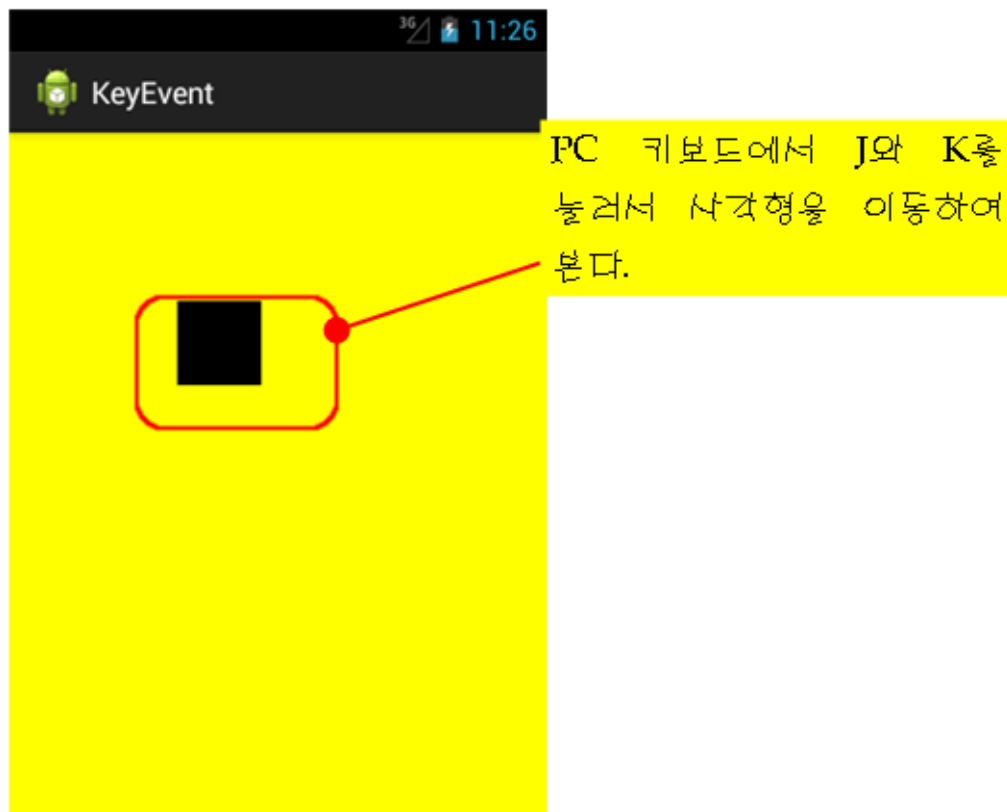
커스텀 버튼

```
public class MainActivity extends Activity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
        final Button button = (Button) findViewById(R.id.button);  
        button.setOnClickListener(new OnClickListener() {  
            public void onClick(View v) {  
                Toast.makeText(getApplicationContext(), "Beep Bop",  
                    Toast.LENGTH_SHORT).show();  
            }  
        });  
    }  
}
```



키이벤트

- 키 이벤트를 받으려면 OnKeyListener 리스너를 구현
- 키를 누르면 사각형이 이동하는 예제를 작성하여 보자.



커스텀 뷰를 정의

```
...
public class KeyEventActivity extends Activity {
    int x = 100, y = 100;
    protected class MyView extends View {
        public MyView(Context context) {
            super(context);
            setBackgroundColor(Color.YELLOW);
        }
        @Override
        protected void onDraw(Canvas canvas) {
            Paint paint = new Paint();
            canvas.drawRect(x, y, x+50, y+50, paint);
        }
    }
}
```

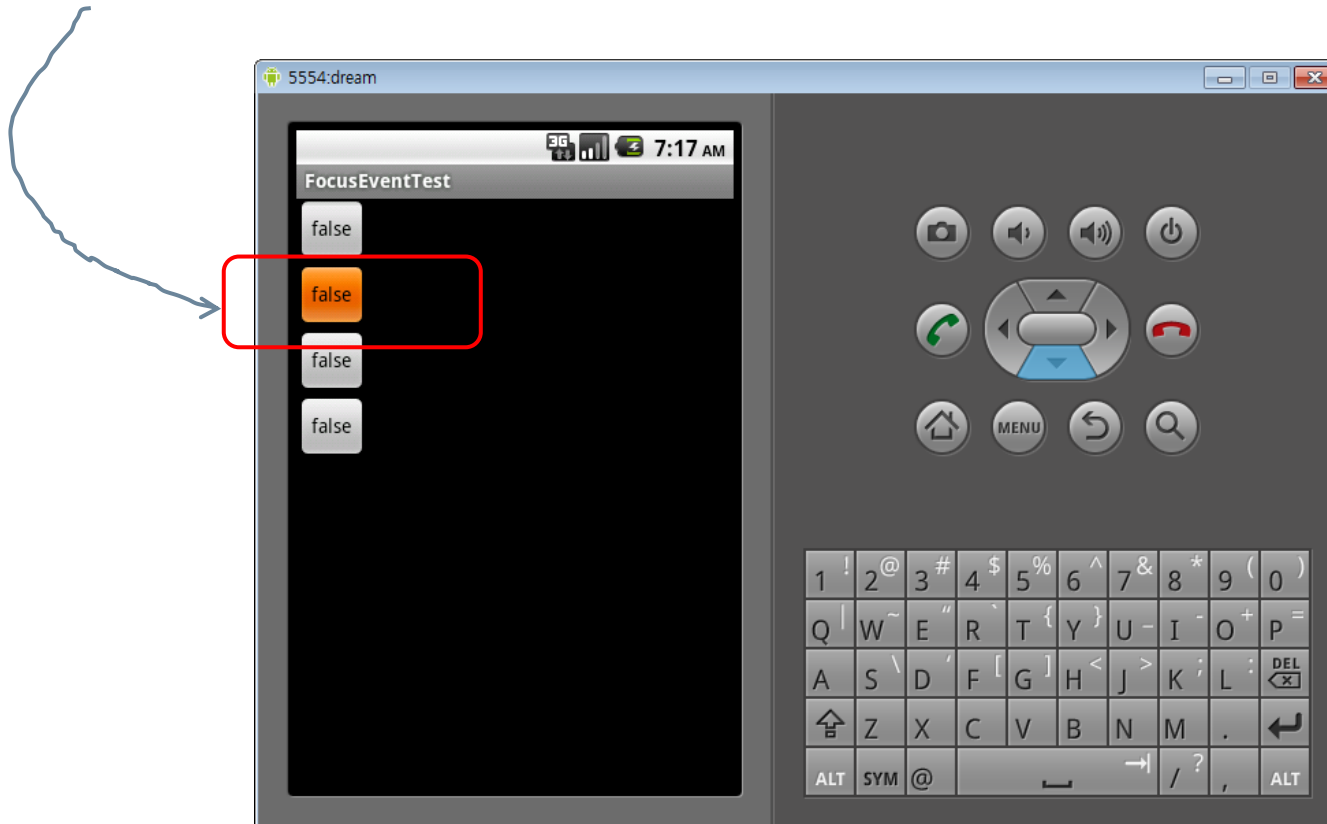
커스텀 뷰를 정의

@Override

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    MyView w = new MyView(this);  
    w.setFocusable(true);  
    w.setFocusableInTouchMode(true);  
    setContentView(w);  
    w.setOnKeyListener(new OnKeyListener() {  
        public boolean onKey(View v, int keyCode, KeyEvent event) {  
            if (event.getAction() == KeyEvent.ACTION_UP) {  
                switch (keyCode) {  
                    case KeyEvent.KEYCODE_J:  
                        x -= 30;  
                        break;  
                    case KeyEvent.KEYCODE_K:  
                        x += 30;  
                        break;  
                }  
                v.invalidate();  
                return true;  
            }  
            return false;  
        }  
    });  
}
```

키 포커스

- 포커스를 가진 위젯 만이 키패드를 통하여 입력을 받을 수 있다.



키 포커스를 위젯에 설정하려면

```
w.setFocusable(true);  
w.setFocusableInTouchMode(true);
```


터치 이벤트

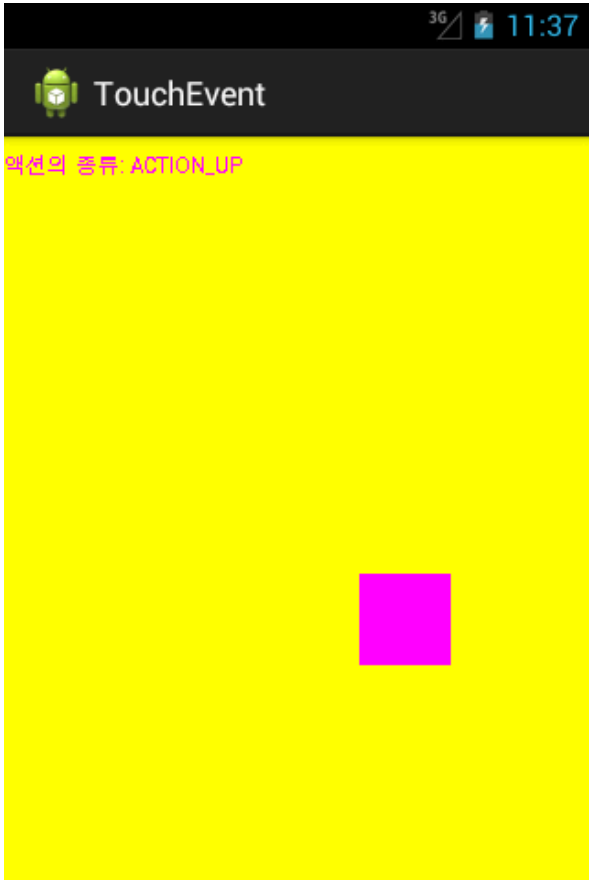
- 일반적으로 커스텀 뷰를 정의하고 onTouchEvent() 재정의

```
class MyView extends View { ...  
    @Override  
    public boolean onTouchEvent(MotionEvent event) {  
        x = (int) event.getX();  
        y = (int) event.getY();  
        ...  
    }  
}
```

터치 이벤트의 종류

액션	설명
ACTION_DOWN	누르는 동작이 시작됨
ACTION_UP	누르고 있다가 <u>떨어</u> 떨 때 발생함
ACTION_MOVE	누르는 도중에 움직임
ACTION_CANCEL	터치 동작이 취소됨
ACTION_OUTSIDE	터치가 현재의 위젯을 벗어남



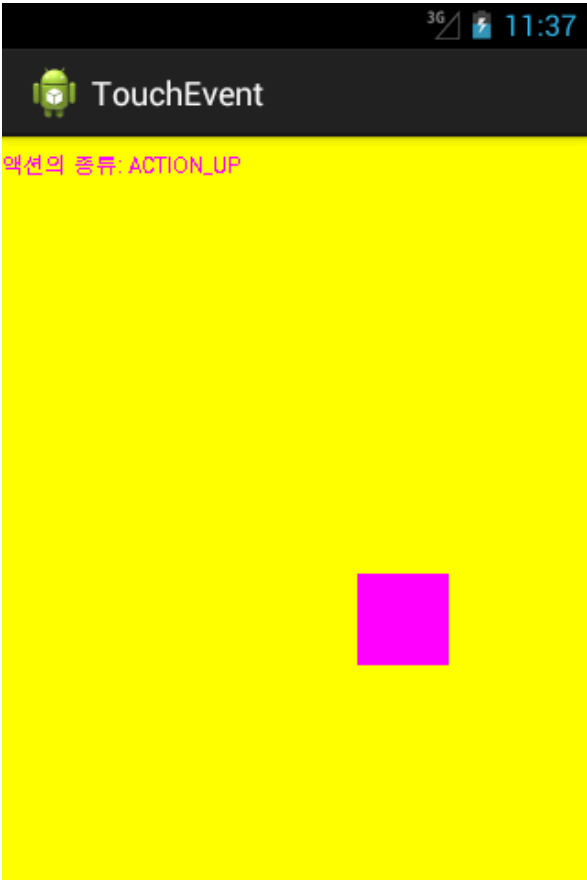


터치 이벤트 예제

```
public class TouchEventActivity extends Activity {  
    protected class MyView extends View {  
        int x = 100, y = 100;  
        String str;  
        public MyView(Context context) {  
            super(context);  
            setBackgroundColor(Color.YELLOW);  
        }  
        @Override  
        protected void onDraw(Canvas canvas) {  
            Paint paint = new Paint();  
            paint.setColor(Color.MAGENTA);  
            canvas.drawRect(x, y, x + 50, y + 50, paint);  
            canvas.drawText("액션의 종류: " + str, 0, 20, paint);  
        }  
    }  
}
```

터치 이벤트 예제

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    x = (int) event.getX();
    y = (int) event.getY();
    if (event.getAction() == MotionEvent.ACTION_DOWN)
        str = "ACTION_DOWN";
    if (event.getAction() == MotionEvent.ACTION_MOVE)
        str = "ACTION_MOVE";
    if (event.getAction() == MotionEvent.ACTION_UP)
        str = "ACTION_UP";
    invalidate();
    return true;
}
}
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    MyView w = new MyView(this);
    setContentView(w);
}
}
```



터치로 곡선 그리기



터치로 곡선 그리기 #1

SingleTouchActivity.java

```
package kr.co.company.singletouch;
```

```
// 소스만 입력하고 Ctrl-Shift-O를 눌러서 import 문장을 자동으로 생성한다.
```

```
public class SingleTouchActivity extends Activity {
```

```
    @Override
```

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

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

```
        setContentView(new SingleTouchView(this, null));
```

```
    }
```

```
}
```

터치로 곡선 그리기 #2

SingleTouchView.java

```
package kr.co.company.singletouch;
```

// 소스만 입력하고 Ctrl-Shift-O를 눌러서 import 문장을 자동으로 생성한다.

```
public class SingleTouchView extends View {  
    private Paint paint = new Paint();  
    private Path path = new Path();
```

```
    public SingleTouchView(Context context, AttributeSet attrs) {  
        super(context, attrs);
```

```
        paint.setAntiAlias(true);  
        paint.setStrokeWidth(10f);  
        paint.setColor(Color.BLUE);  
        paint.setStyle(Paint.Style.STROKE);  
        paint.setStrokeJoin(Paint.Join.ROUND);
```

```
    }
```

```
@Override
```

```
protected void onDraw(Canvas canvas) {  
    canvas.drawPath(path, paint);  
}
```

선분을 매끄럽게 그리기 위하여 엔티 에일리어싱을 설정한다.

선분의 두께를 10으로 한다.

현재까지의 경로를 모두 그린다.

터치로 곡선 그리기 #3

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    float eventX = event.getX();
    float eventY = event.getY();

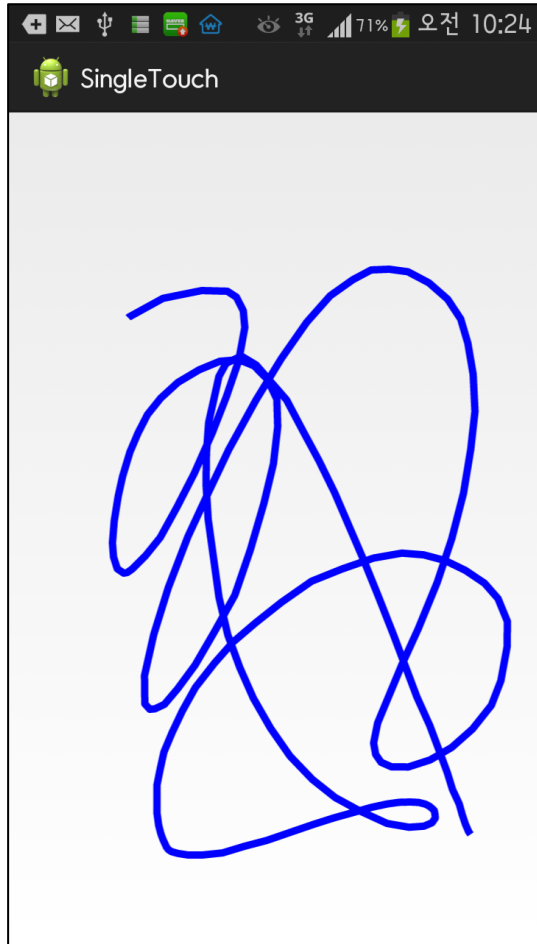
    switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            path.moveTo(eventX, eventY);
            return true;
        case MotionEvent.ACTION_MOVE:
            path.lineTo(eventX, eventY);
            break;
        case MotionEvent.ACTION_UP:
            break;
        default:
            return false;
    }

    invalidate();
    return true;
}
```

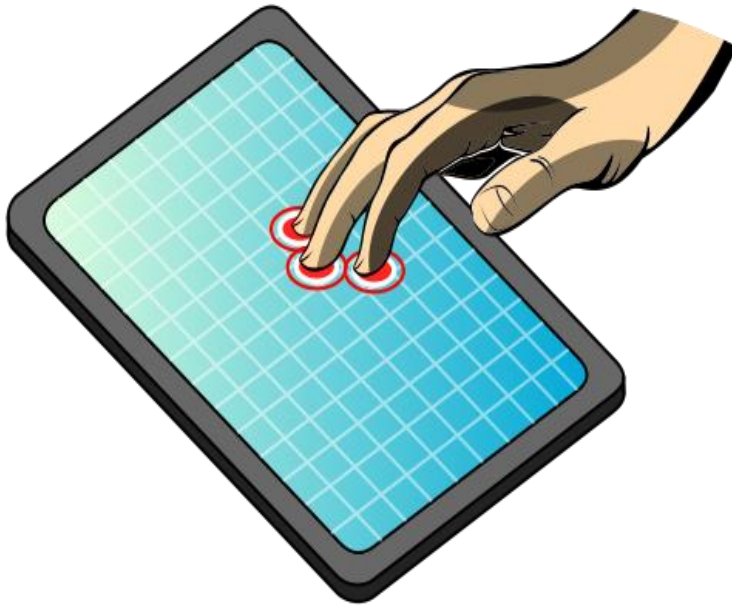
마우스가 터치된 위치를 얻는다.

터치가 내려지면 경로에 위치를 저장한다.

터치가 떴어지면 경로에 직선그리기를 저장한다.



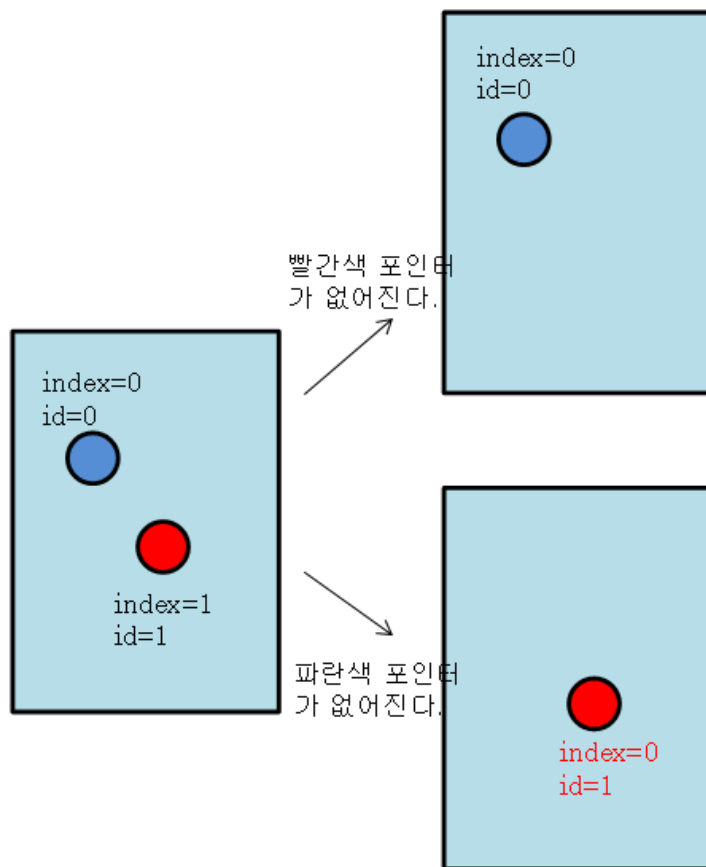
- 여러 개의 손가락을 이용하여 화면을 터치하는 것으로 이미지를 확대/축소할 때 많이 사용된다.



터치 이벤트

- ACTION_DOWN – 화면을 터치하는 첫 번째 포인터에 대하여 발생한다. 제스처 인식이 시작된다. 첫 번째 터치는 항상 MotionEvent에서 인덱스 0번에 저장된다.
- ACTION_POINTER_DOWN – 첫 번째 포인터 이외의 포인터에 대하여 발생된다. 포인터 데이터는 `getActionIndex()`이 반환하는 인덱스에 저장된다.
- ACTION_MOVE – 화면을 누르면서 이동할 때 발생한다.
- ACTION_POINTER_UP – 마지막 포인터가 아닌 다른 포인터가 화면에서 없어지면 발생된다.
- ACTION_UP – 화면을 떠나는 마지막 포인터에 대하여 발생된다.

인덱스와 아이디



파란색 포인터가 없어지는 경우에 index와 id가 달라집니다.



터치된 위치에 원을 그리는 예제

MultiTouchActivity.java

```
package kr.co.company.multitouch;
```

```
// 소스만 입력하고 Ctrl-Shift-O를 눌러서 import 문장을 자동으로 생성한다.
```

```
public class MultiTouchActivity extends Activity {
```

```
    @Override
```

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

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

```
        setContentView(new MultiTouchView(this, null));
```

```
    }
```

```
}
```

액티비티의 화면을
MultiTouchView 객체로 설정한다.

MultiTouchView.java

```
package kr.co.company.multitouch;
```

```
// 소스만 입력하고 Ctrl-Shift-O를 눌러서 import 문장을 자동으로 생성한다.
```

```
public class MultiTouchView extends View {
```

```
    private static final int SIZE = 60;
```

```
    final int MAX_POINTS = 10;
```

```
    float[] x = new float[MAX_POINTS];
```

```
    float[] y = new float[MAX_POINTS];
```

```
    boolean[] touching = new boolean[MAX_POINTS];
```

```
    private Paint mPaint;
```

```
    public MultiTouchView(Context context, AttributeSet attrs) {
```

```
        super(context, attrs);
```

```
        initView();
```

```
    }
```

```
    private void initView() {
```

```
        mPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
```

```
        mPaint.setColor(Color.BLUE);
```

```
        mPaint.setStyle(Paint.Style.FILL_AND_STROKE);
```

```
    }
```

최대 10개 포인트의 위치와
상태를 저장할 수 있다.

@Override

```
public boolean onTouchEvent(MotionEvent event) {
```

```
    int index = event.getActionIndex();  
    int id = event.getPointerId(index);  
    int action = event.getActionMasked();
```

인덱스로부터 포인터의 아이디를 구한다.

```
    switch (action) {
```

```
        case MotionEvent.ACTION_DOWN:
```

```
        case MotionEvent.ACTION_POINTER_DOWN:
```

```
            x[id] = (int) event.getX(index);
```

```
            y[id] = (int) event.getY(index);
```

```
            touching[id] = true;
```

```
            break;
```

```
        case MotionEvent.ACTION_MOVE:
```

```
            break;
```

```
        case MotionEvent.ACTION_UP:
```

```
        case MotionEvent.ACTION_POINTER_UP:
```

```
        case MotionEvent.ACTION_CANCEL:
```

```
            touching[id] = false;
```

```
            break;
```

```
    }
```

```
    invalidate();
```

```
    return true;
```

```
}
```

화면이 터치되면 위치를 계산하여 배열에 저장한다.
touching[] 배열에 true를 저장하여서 현재 터치가 되어 있다는 것을 표시한다.

처리가 종료되었음을 저장한다.

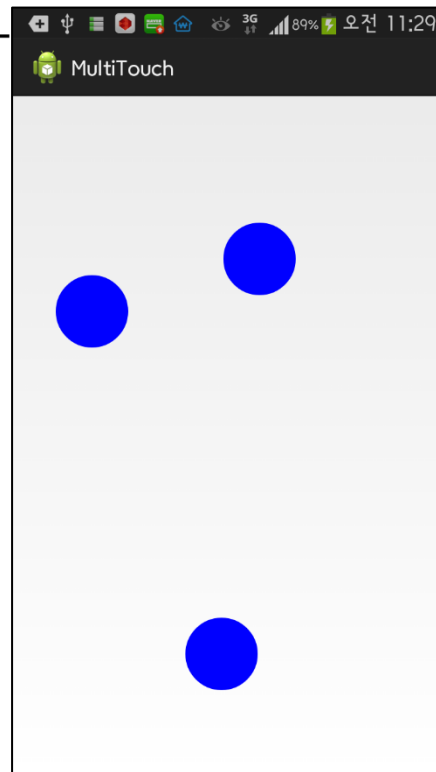
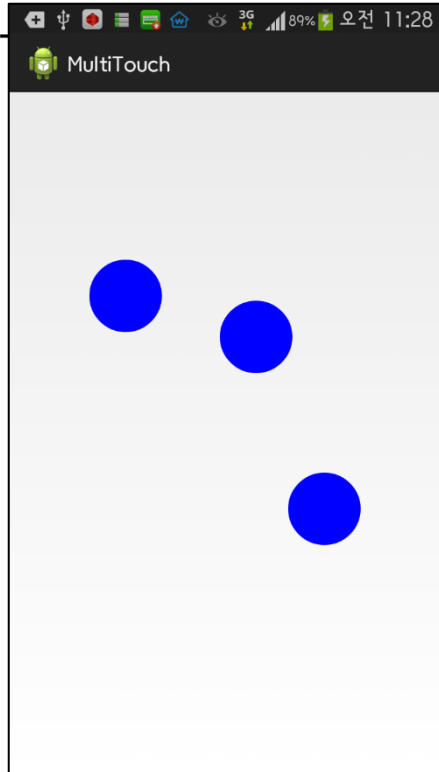
@Override

```
protected void onDraw(Canvas canvas) {  
    super.onDraw(canvas);
```

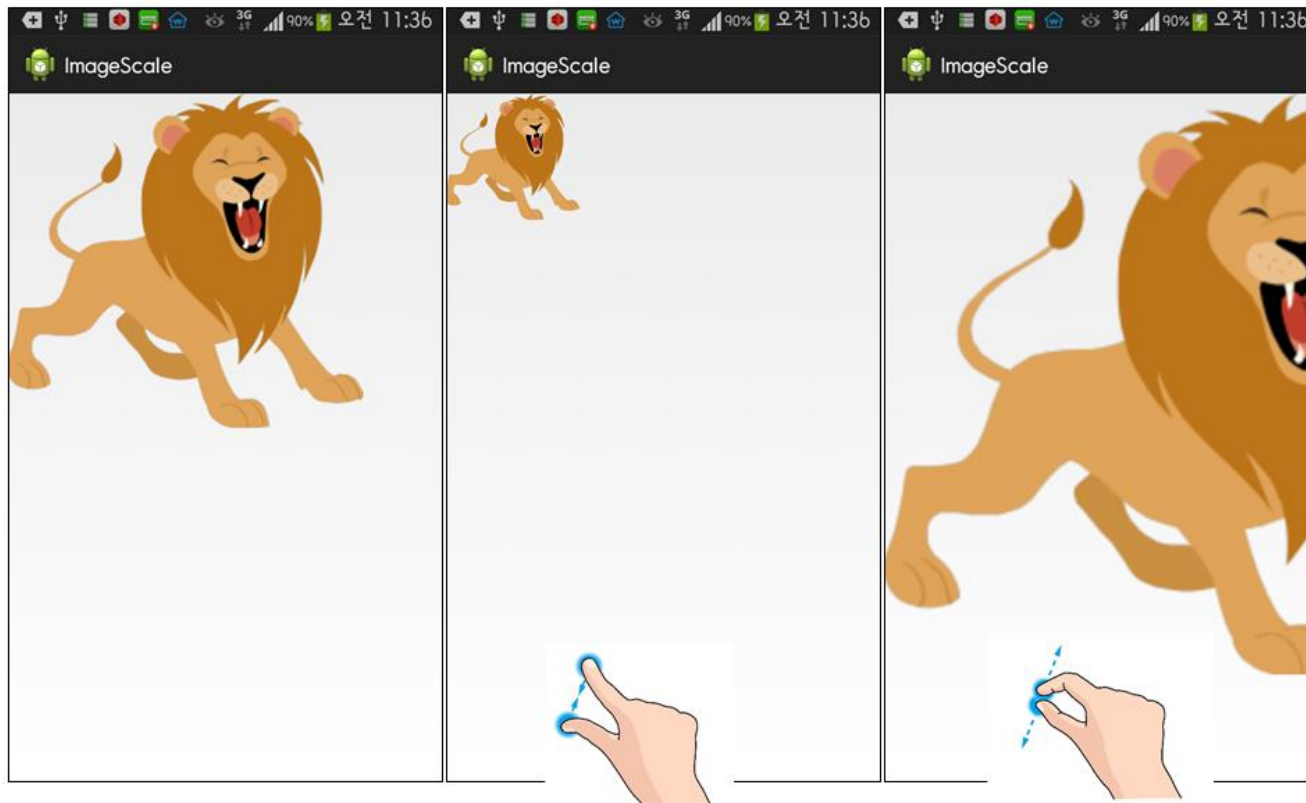
```
    for (int i = 0; i < MAX_POINTS; i++) {  
        if (touching[i]) {  
            canvas.drawCircle(x[i], y[i], SIZE, mPaint);  
        }  
    }  
}
```

현재 터치되어 있는 포인트
위치에 원을 그린다.

```
}
```



핀치줌 구현



액티비티 정의

ImageScaleActivity.java

```
package kr.co.company.imagescale;
```

```
// 소스만 입력하고 Alt+Enter를 눌러서 import 문장을 자동으로 생성한다.
```

```
public class ImageScaleActivity extends ActionBarActivity {
```

```
    @Override
```

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

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

```
        setContentView(new MyImageView(this));
```

```
    }
```

```
}
```

뷰 정의

```
public class MyImageView extends View {  
    private Drawable image;  
    private ScaleGestureDetector gestureDetector;  
    private float scale = 1.0f;  
  
    public MyImageView(Context context) {  
        super(context);  
        image = context.getResources().getDrawable(R.drawable.lion);  
        setFocusable(true);  
        image.setBounds(0, 0, image.getIntrinsicWidth(),  
            image.getIntrinsicHeight());  
        gestureDetector = new ScaleGestureDetector(context, new ScaleListener());  
    }  
  
    @Override  
    protected void onDraw(Canvas canvas) {  
        super.onDraw(canvas);  
        canvas.save();  
        canvas.scale(scale, scale);  
        image.draw(canvas);  
        canvas.restore();  
    }  
  
    @Override  
    public boolean onTouchEvent(MotionEvent event) {  
        gestureDetector.onTouchEvent(event);  
        invalidate();  
        return true;  
    }  
}
```

제스처 인식기 객체를 생성
한다.

캔버스에 선축 연산을 적용
한다. 좀 더 자세한 내용은
다음 장을 참조한다.

제스처 인식기의 터치 이벤
트 처리 메소드를 호출해준
다.

뷰 정의

```
private class ScaleListener extends
    ScaleGestureDetector.SimpleOnScaleGestureListener {
    @Override
    public boolean onScale(ScaleGestureDetector detector) {
        scale *= detector.getScaleFactor();

        if (scale < 0.1f)
            scale = 0.1f;
        if (scale > 10.0f)
            scale = 10.0f;

        invalidate();
        return true;
    }
}
```

신축 연산이 감지되었으면 호출된다.