# CHAP 6. 이벤트 처리(3)

# 터치로 곡선 그리기




© 2012 생능출판사 All rights reserved

# 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));
    }

}
```



© 2012 생능출판사 All rights reserved

# 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 앤티 에일리어싱을 설정한다.

선분의 두께를 10으로 한다.

현재까지의 경로를 모두 그린다.



© 2012 생능출판사 All rights reserved

# 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;
  }
}
```

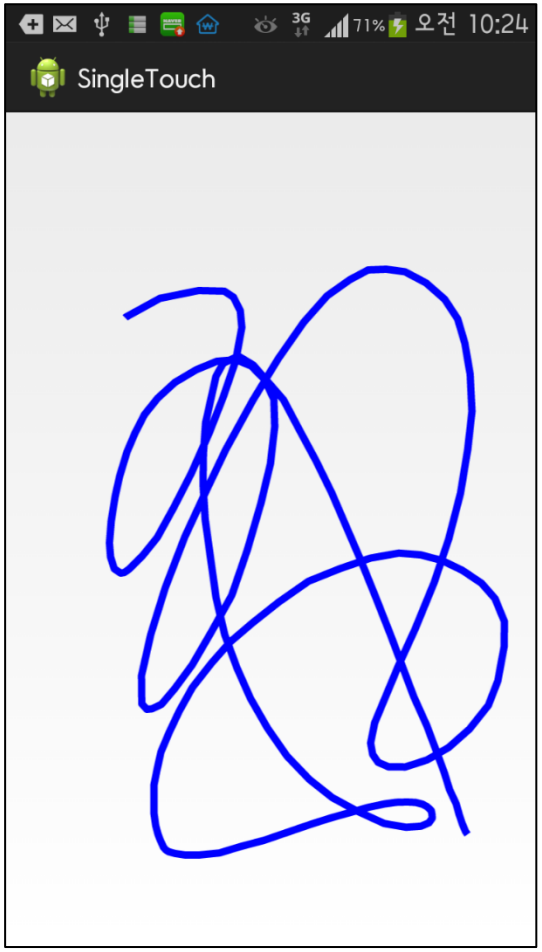
마우스가 터치된 위치를 얻는다.

터치가 눌려지면 경로에 위치를 저장한다.

터치가 떼어지면 경로에 직선그리기를 저장한다.

© 2012 생능출판사 All rights reserved

# 실행결과






© 2012 생능출판사 All rights reserved

# 멀티 터치

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



© 2012 생능출판사 All rights reserved

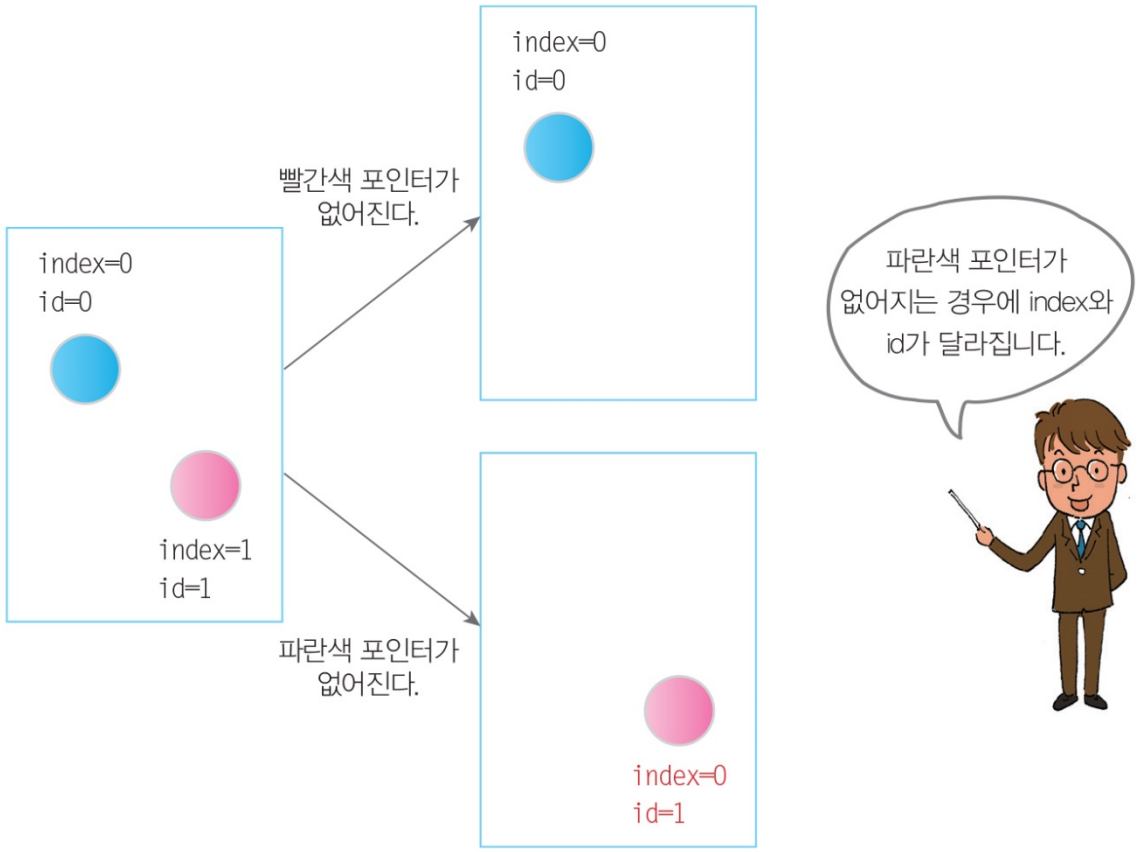
# 터치 이벤트

- 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.



© 2012 생능출판사 All rights reserved

# 인덱스와 아이디






© 2012 생능출판사 All rights reserved

# 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.



© 2012 생능출판사 All rights reserved

*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 포인터의 위치와 상태를 저장할 수 있다.



© 2012 생능출판사 All rights reserved

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

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

    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;
}
```

인덱스로부터 포인터의 아이디를 구한다.

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

처리가 종료되었음을 저장한다.

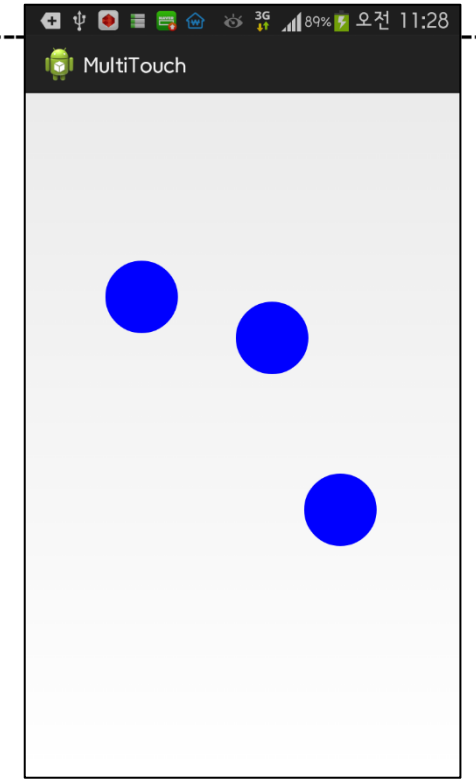


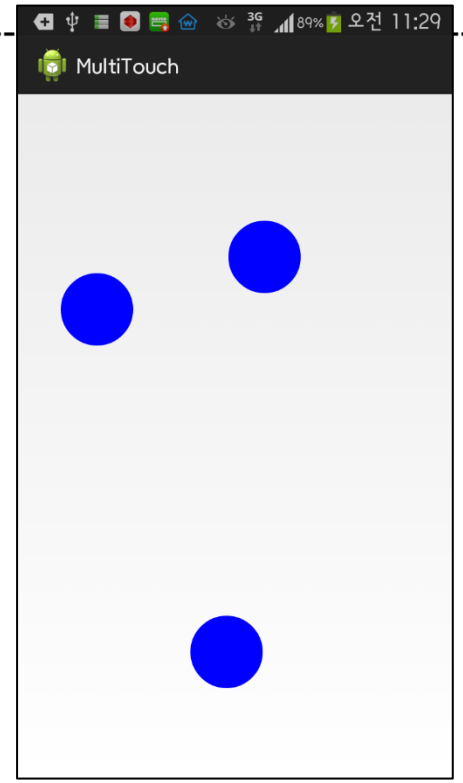
© 2012 생능출판사 All rights reserved

```
@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 포인터 위치에 원을 그린다.

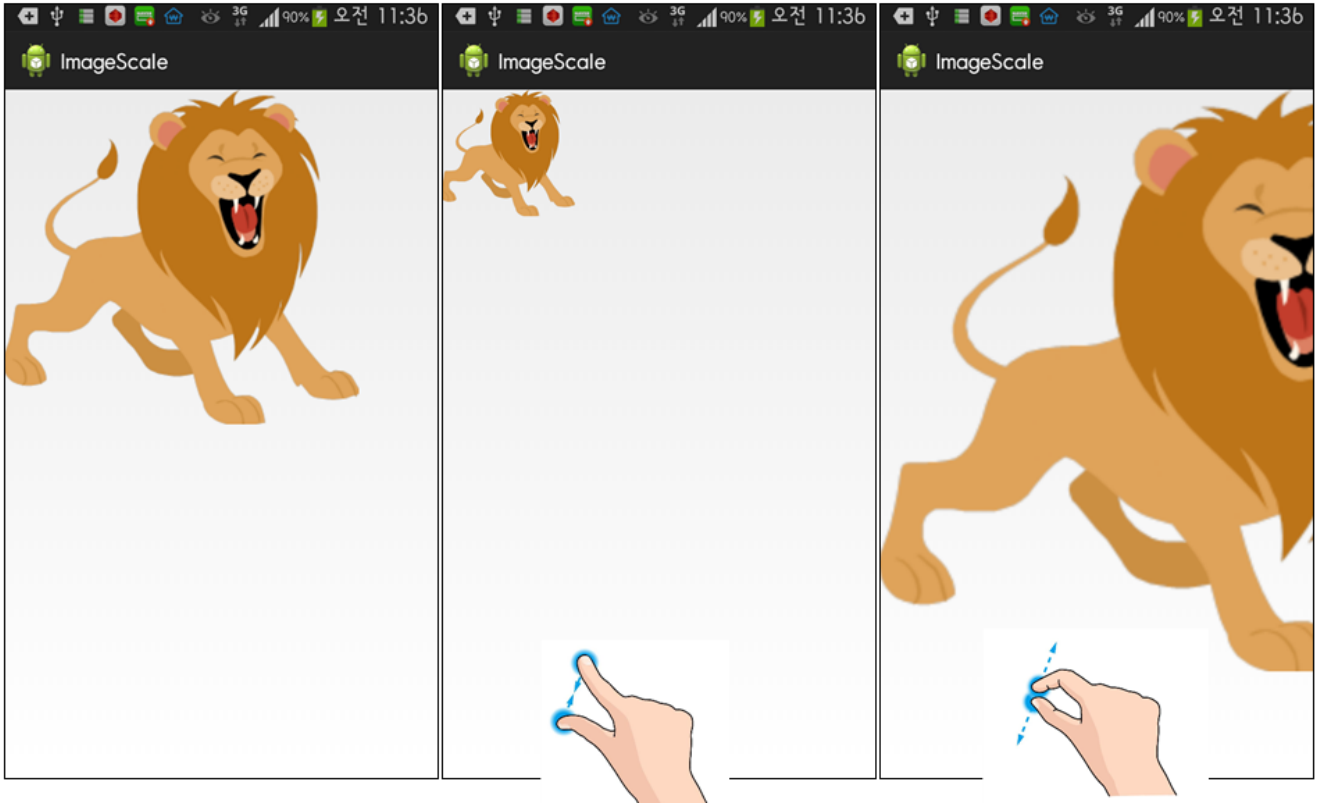





© 2012 생능출판사 All rights reserved

# 핀치줌 구현






© 2012 생능출판사 All rights reserved

# 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));
    }
}
```



© 2012 생능출판사 All rights reserved

# 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 생성한다.

캔버스에 신축 연산을 적용한다. 좀 더 자세한 내용은 다음 장을 참조한다.

제스처 인식기의 터치 이벤트 처리 메소드를 호출해준다.

# 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.

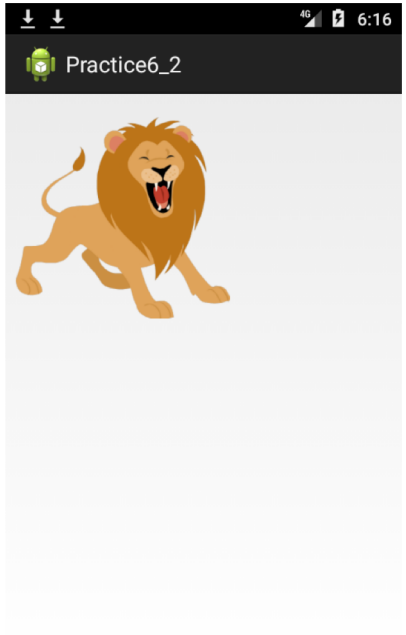
---

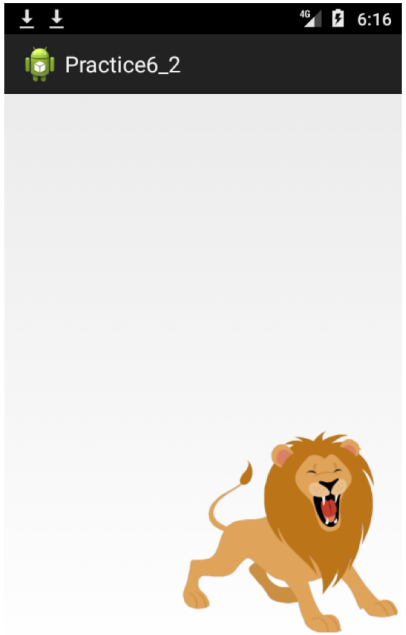


© 2012 생능출판사 All rights reserved

# 실습 2

- lion.png 이미지 move
  - 한 손가락의 위치를 따라 lion.png 이미지가 움직이도록 구현하시오.
    - canvas.translate()을 활용하시오
  - 두 손가락은 확대/축소가 되도록 구현하시오.







© 2012 생능출판사 All rights reserved