Multi-Touch in Android

Jianjun Chen (Jianjun.Chen@xjtlu.edu.cn)

Category	Methods	Description
Creation	Constructors	There is a form of the constructor that are called when the view is created from code and a form that is called when the view is inflated from a layout file. The second form should parse and apply any attributes defined in the layout file.
	`onFinishInflate()`	Called after a view and all of its children has been inflated from XML.
Layout	`onMeasure(int, int)`	Called to determine the size requirements for this view and all of its children.
	`onLayout(boolean, int, int, int, int)`	Called when this view should assign a size and position to all of its children.
	`onSizeChanged(int, int, int, int)`	Called when the size of this view has changed.
Drawing	`onDraw(android.graphics.Canvas)`	Called when the view should render its content.
Event processing	`onKeyDown(int, KeyEvent)`	Called when a new hardware key event occurs.
	`onKeyUp(int, KeyEvent)`	Called when a hardware key up event occurs.
	`onTrackballEvent(MotionEvent)`	Called when a trackball motion event occurs.
	`onTouchEvent(MotionEvent)`	Called when a touch screen motion event occurs.
Focus	`onFocusChanged(boolean, int, android.graphics.Rect)`	Called when the view gains or loses focus.
	`onWindowFocusChanged(boolean)`	Called when the window containing the view gains or loses focus.
Attaching	`onAttachedToWindow()`	Called when the view is attached to a window.
	`onDetachedFromWindow()`	Called when the view is detached from its window.
	`onWindowVisibilityChanged(int)`	Called when the visibility of the window containing the view has changed.

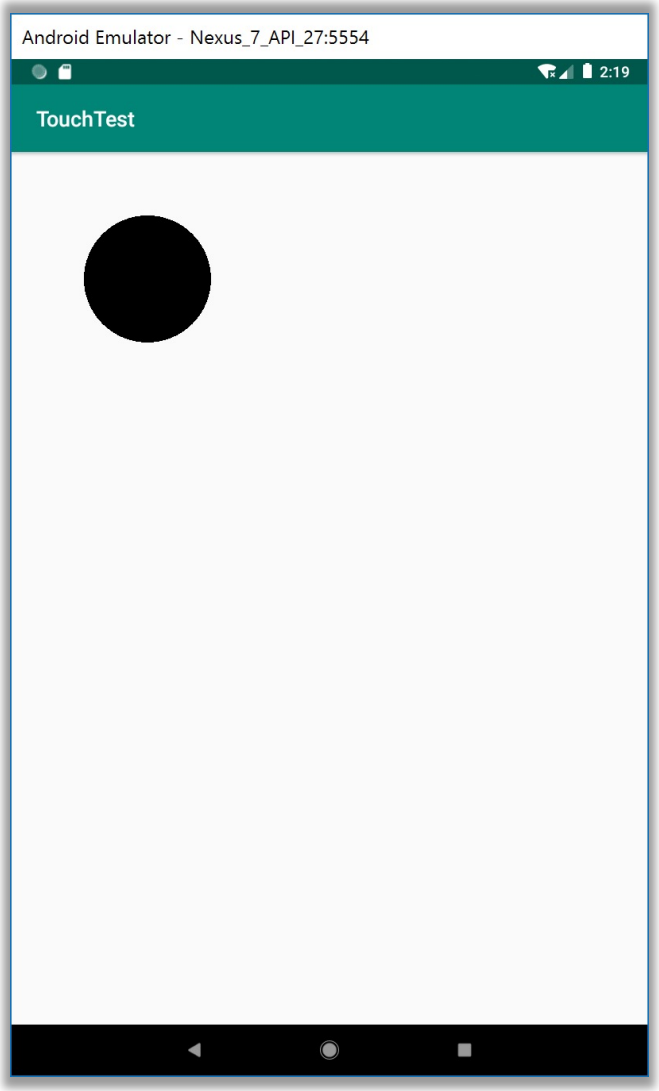
Single-Touch

Touch events and project structure

Single-Touch App

- Let's start from a single-touch app to explain the touch event mechanisms.

- In the next example, we will create a `View`.
  - whenever this `View` is touched by the user, it will draw a large black dot at the touched position.



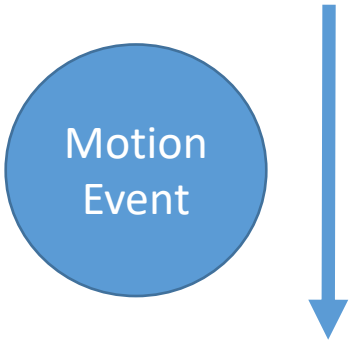
```java
public class TouchView extends View {
    private float pX;
    private float pY;

    public TouchView (Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    protected void onDraw(Canvas canvas) {
        super.onDraw(canvas);
        canvas.drawCircle(pX, pY,
                          canvas.getWidth() / 10, new Paint());
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        pX = event.getX();
        pY = event.getY();
        invalidate();
        return true;
    }
}
```

Class for touch, mouse events.

Inform this `View` that it should update itself. `onDraw()` will be called by the system.

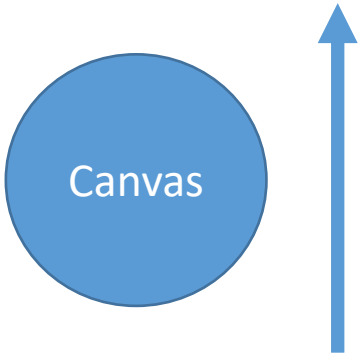
Flow of This Example

User touches the screen, or moves his finger on the screen. The system generates the `MotionEvent`.




```
onTouchEvent(      ) {
    View.invalidate();
}
```

The Android system finds a suitable time for the UI thread to update this `View`



`onDraw()` is called. A circle is drawn on the screen.

Question

- From the previous analysis, we can see that the `TouchView`'s UI update is driven by the touch events.

- That is, while the screen is being touched, the system keeps calling `onTouchEvent()`, which then leads to UI updates through `invalidate()`

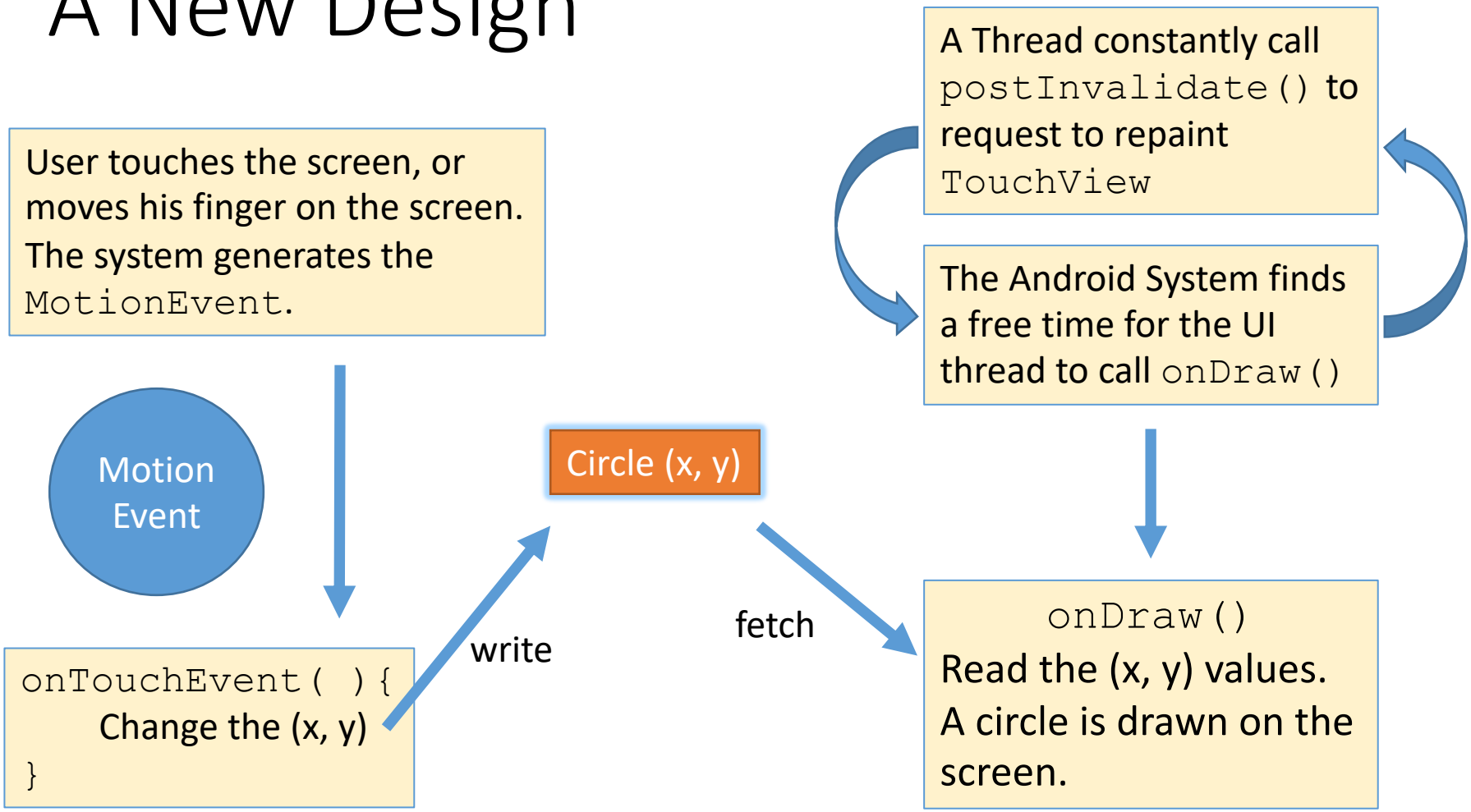
Question

- This design will be problematic if we have another object that moves by itself in this View.
  - Such as a bouncing ball.

- Solution: change the design of our `View` so that it update itself at a fixed frequency.
  - E.g. at 60 frames per second (FPS).

- We should use a thread.

A New Design



The bouncing ball thread is not included here due to the screen space.

The Thread

Use inner class here, so that we can access the `TouchView` object easily

```
public class TouchView extends View {
    class MyThread extends Thread {
        @Override
        public void run() {
            while(!this.isInterrupted()) {
                try {
                    TouchView.this.postInvalidate();
                    Thread.sleep(16); // around 60fps
                } catch (InterruptedException e) {
                    return;
                }
            }
        }
    }
...
```

The Rest of the TouchView

```
private float pX;
private float pY;
MyThread t;

public TouchView (Context context, AttributeSet attrs) {
    super(context, attrs);
}

@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    canvas.drawCircle(pX, pY, canvas.getWidth() / 10, new Paint());
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    pX = event.getX();
    pY = event.getY();
    return true;
}
```

Starting MyThread in Activity

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    TouchView touchView = findViewById(R.id.touchView);
    touchView.t = touchView.new MyThread();
    touchView.t.start();
}
```

- Question: If the app is put into back stack , will `MyThread` keep running?

Starting MyThread in Activity

- Answer: Threads do run at background even when the activity is paused.
  - `postInvalidate()` will still be called.
  - but `View`'s `onDraw()` will not run by the system when the app is paused.

```
while(!this.isInterrupted()) {
    try {
        TouchView.this.postInvalidate();
        Thread.sleep(16); // around 60fps
    } catch (InterruptedException e) {
        return; // stop if this thread is interrupted
    }
}
```

Controlling MyThread

- You can also control the `View` update by explicitly starting/stopping `MyThread`.
  - However, keep in mind that a stopped Thread CANNOT be started again using `Thread.start()`.
  - You need to re-create the thread object and start again

- Start:

```
// a stopped thread cannot be started again, re-create it
touchView.t = touchView.new MyThread();
touchView.t.start();
```

- Stop:

```
TouchView touchView = findViewById(R.id.touchView);
touchView.t.interrupt();
```

Thread Interruption

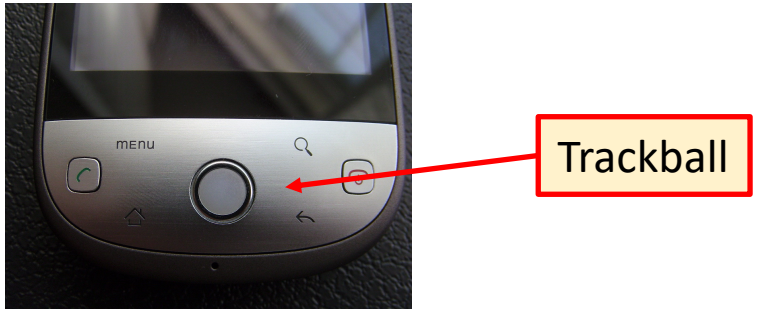
- `Interrupt()` can stop threads under the effect of `sleep()`.
  - https://docs.oracle.com/javase/tutorial/essential/concurrency/interrupt.html

```
while(!this.isInterrupted()) {// stop if this thread is interrupted
    try {
        TouchView.this.postInvalidate();
        Thread.sleep(16); // around 60fps
    } catch (InterruptedException e) {
        return; // stop if this thread is interrupted
    }
}
```

Must return, otherwise it will loop again

MotionEvent & Multi-Touch

MotionEvent



- A `MotionEvent` object contains the information about the movements of finger, mouse, pen and trackball.
  - Every time a new finger (pointer) touches the screen. Android will assign a pointer index and a pointer ID to it.
  - Every time a finger lifts from the screen, that ID and index will be released.

- Each event describes an action happend. Based on the action type, we can carry out different operations.

Getting the Action Type

- Prior to Android 2.2, only single-touch gestures are supported, action type is obtained by calling `getAction()`.
- Android 2.2 introduced multi-touch, we should now stick to `getActionMasked()`.

- When touching the screen with a single finger, both functions will return the same result.

Action Types Returned

- `MotionEvent.ACTION_DOWN`:
  - Issued when the first pointer touched the screen.
- `MotionEvent.ACTION_UP`:
  - Issued when the last pointer leaves the screen.
- `MotionEvent.ACTION_CANCEL`:
  - Issued when the user cancelled the gesture.
  - http://stackoverflow.com/questions/11960861/what-causes-a-motionevent-action-cancel-in-android

Action Types Returned

- `MotionEvent.ACTION_POINTER_DOWN`:
  - Another finger (A non-primary pointer) touched the screen.
- `MotionEvent.ACTION_POINTER_UP`:
  - A non-primary pointer lifted up.
- `MotionEvent.ACTION_MOVE`:
  - Issued when the user is moving his finger(or other press gestures from input devices like pen, mouse …).

Identifying Pointers

Each pointer is associated with an **index** and an ID:

- **Index**: `MotionEvent` stores pointers in an array, the index of a pointer is its index in this array.
  - The index of a pointer can change from one event to the next.
  - To get the index that triggered the motion event, call `MotionEvent.getActionIndex()`
  - The index may be used with `getPointerId(int)`, `getX(int)`, `getY(int)`, `getPressure(int)`, and `getSize(int)` to get related information.

Identifying Pointers

Each pointer is associated with an index and an **ID**:

- **ID**: This ID stays the same as long as the pointer stays on the screen.
  - If you want to track a certain finger, you should track the ID instead of the index.
  - To get the ID of that pointer, call `MotionEvent.getPointerID(int pt_index)`

Pointer Index

MotionEvent 1

	ID	Index
Finger 1	0	0
Finger 2	1	1

MotionEvent 2

	ID	Index
Finger 1	0	1
Finger 2	1	0

The Example App

- In the next example, we will create another customised View that reacts to multi-touch events.

- Whenever a new finger touches the screen, a big dot will be shown on the touched part of the screen.
  - Each dot will have different colours.

```java
public class MultiTouchView extends View {
    class AnimationThread extends Thread {
        @Override
        public void run() {
            while (!this.isInterrupted()) {
                MultiTouchView.this.postInvalidate();
                try {
                    Thread.sleep(16);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                    return;
                }
            }
        }
    }

    public AnimationThread t;

    public MultiTouchView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }
...
```

Similar Thread and Constructor

```
...
private SparseArray<PointF> pointers = new SparseArray<PointF>();

@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    Paint paint = new Paint();
    final int[] colors = {Color.BLUE, Color.GREEN,
                          Color.BLACK, Color.CYAN, Color.GRAY};
    for (int id = 0; id < pointers.size(); id++) {
        PointF point = pointers.valueAt(id);
        if (point != null) {
            paint.setColor(colors[id % 5]);
            canvas.drawCircle(point.x, point.y, 80, paint);
            paint.setTextAlign(Paint.Align.CENTER);
            paint.setTextSize(20);
            canvas.drawText("ID: " + id,
                    point.x, point.y - 100, paint);
        }
    }
}
...
```

`SparseArrays` map integers to objects.
You can treat it as `HashMap<Integer, PointF>`

`onDraw()` and the points

```
...
@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getActionMasked()) {
        case MotionEvent.ACTION_DOWN:
        case MotionEvent.ACTION_POINTER_DOWN:
            int pointerIndex = event.getActionIndex();
            int pointerID = event.getPointerId(pointerIndex);
            PointF f = new PointF();
            f.x = event.getX(pointerIndex);
            f.y = event.getY(pointerIndex);
            pointers.put(pointerID, f);
            break;
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_POINTER_UP:
        case MotionEvent.ACTION_CANCEL:
            pointers.remove(
                        event.getPointerId(event.getActionIndex())
                    );
            break;
...
```

`onTouchEvent()` first half

```java
@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getActionMasked()) {
    ...
        case MotionEvent.ACTION_MOVE:
            for (int i = 0; i < event.getPointerCount(); i++) {
                PointF p = pointers.get(event.getPointerId(i));
                if (p != null) {
                    p.x = event.getX(i);
                    p.y = event.getY(i);
                }
            }
    } // end of switch
    return true;
} // end of onTouchEvent()
```

`onTouchEvent()` second half


无服务
清晨5:02
MultiTouch

getActionMasked()

Action is stored in the last 8 digits.
Pointer index is stored in the first 8 digits.

```
public static final int ACTION_MASK = 0xff;
public static final int ACTION_POINTER_INDEX_MASK = 0xff00;
public static final int ACTION_POINTER_INDEX_SHIFT = 8;

public final int getAction() {
    return mAction;
}
public final int getActionMasked() {
    return mAction & ACTION_MASK;
}
public final int getActionIndex() {
    return (mAction & ACTION_POINTER_INDEX_MASK)
            >> ACTION_POINTER_INDEX_SHIFT;
}
```

Gets the pointer index

This is just a discussion about the implementation of the action system,
This is the source code for `getActionMasked()`

Lab Task 1

- Get familiar with the solution of the last lab. I have already uploaded it.
  - The solution has multi-touch support, so check the code carefully.
  - MAKE SURE you can implement that by yourself!

Lab Task 2

- Draw a bouncing ball (Not controlled by you) that moves slowly.
- When you touch the screen, draw square(s) that follow your finger(s).
- If the bouncing ball enters into the square, accelerate it towards its current moving direction
  - It should be a proper acceleration, using the equation in the physics, not just suddenly increasing the speed.
  - $v = a * t$
  - You can decide the acceleration factor.
- If the ball leaves the square, slowly restore its original speed.
- !!Make sure you can implement this app!!