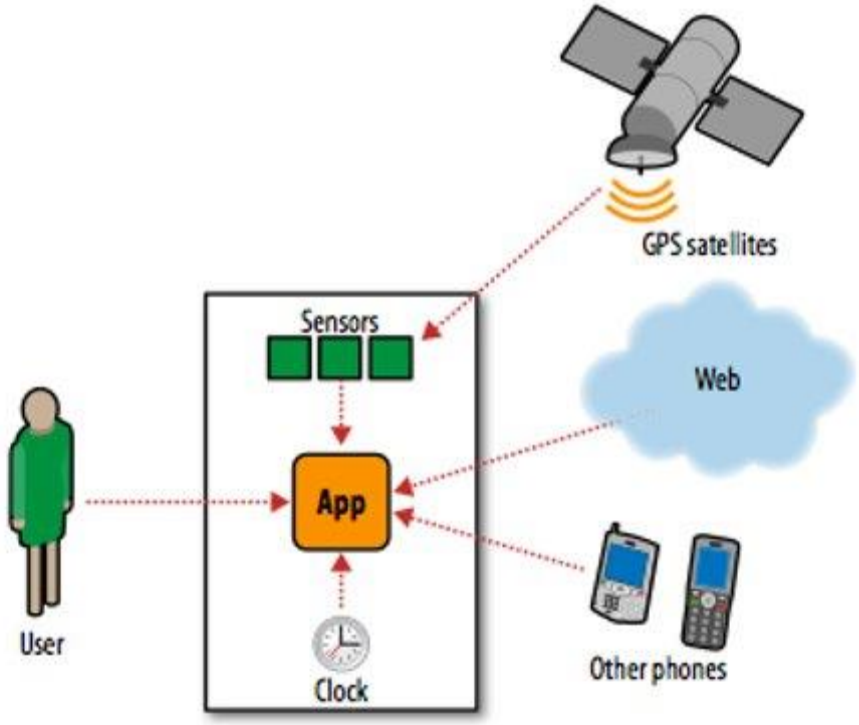
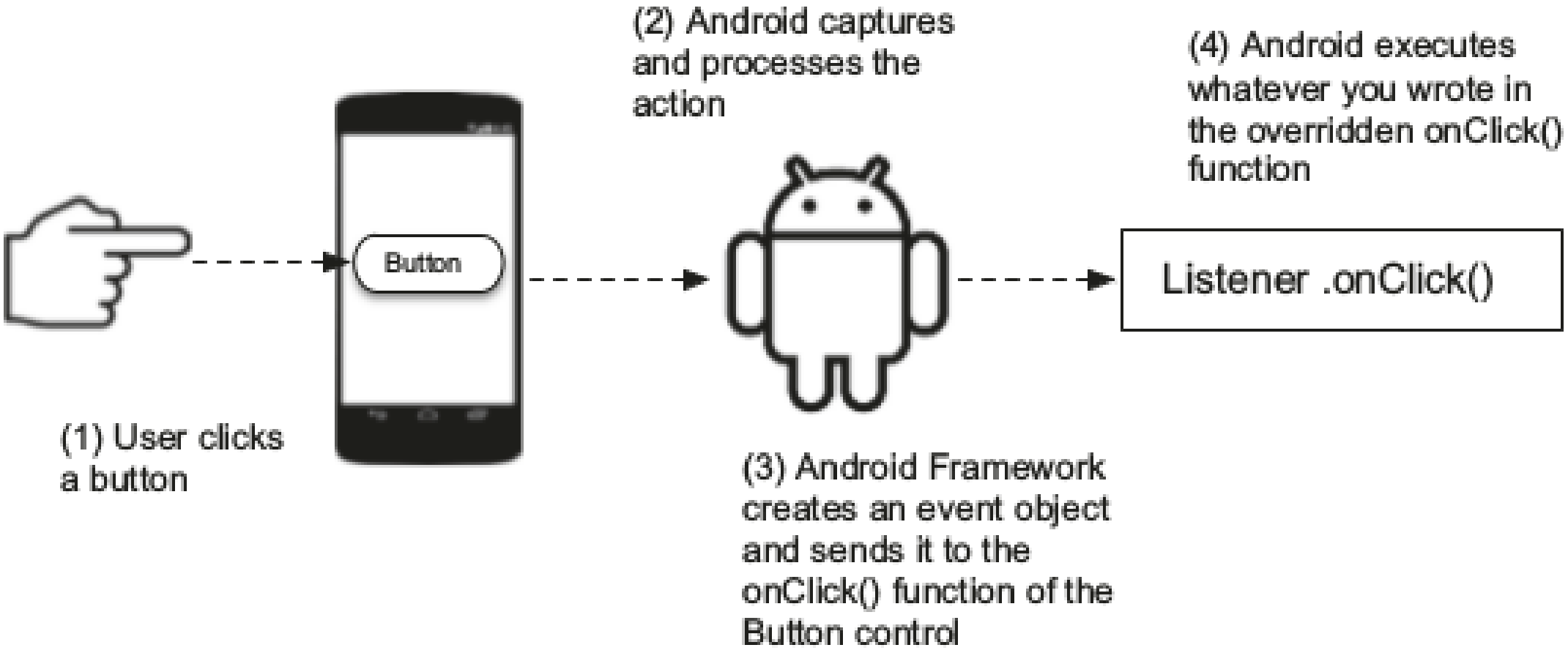
Events

Event driven programming

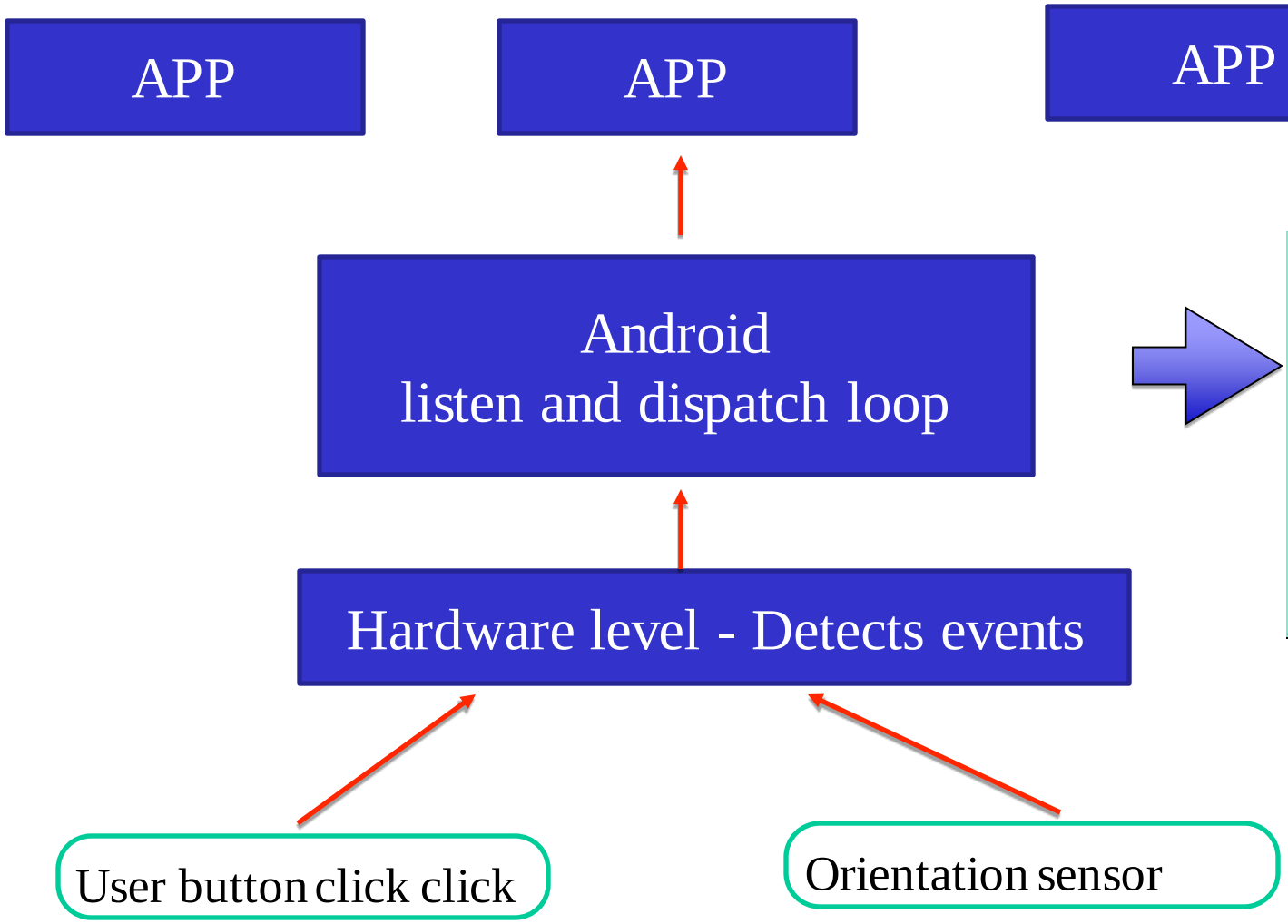
- A type of programming where flow of control is determined by events.
- Events:
  - Physical occurrences in the real world
  - User actions (button click, menu selection)
  - Messages (text message, phone call)
  - Sensor inputs (GPS, orientation change)
  - Program actions (Display, outgoing message)
- Event handlers: software objects (blocks) that we can program to take action in response to events.



Simplified event handling model



Events processing



App must register their handlers

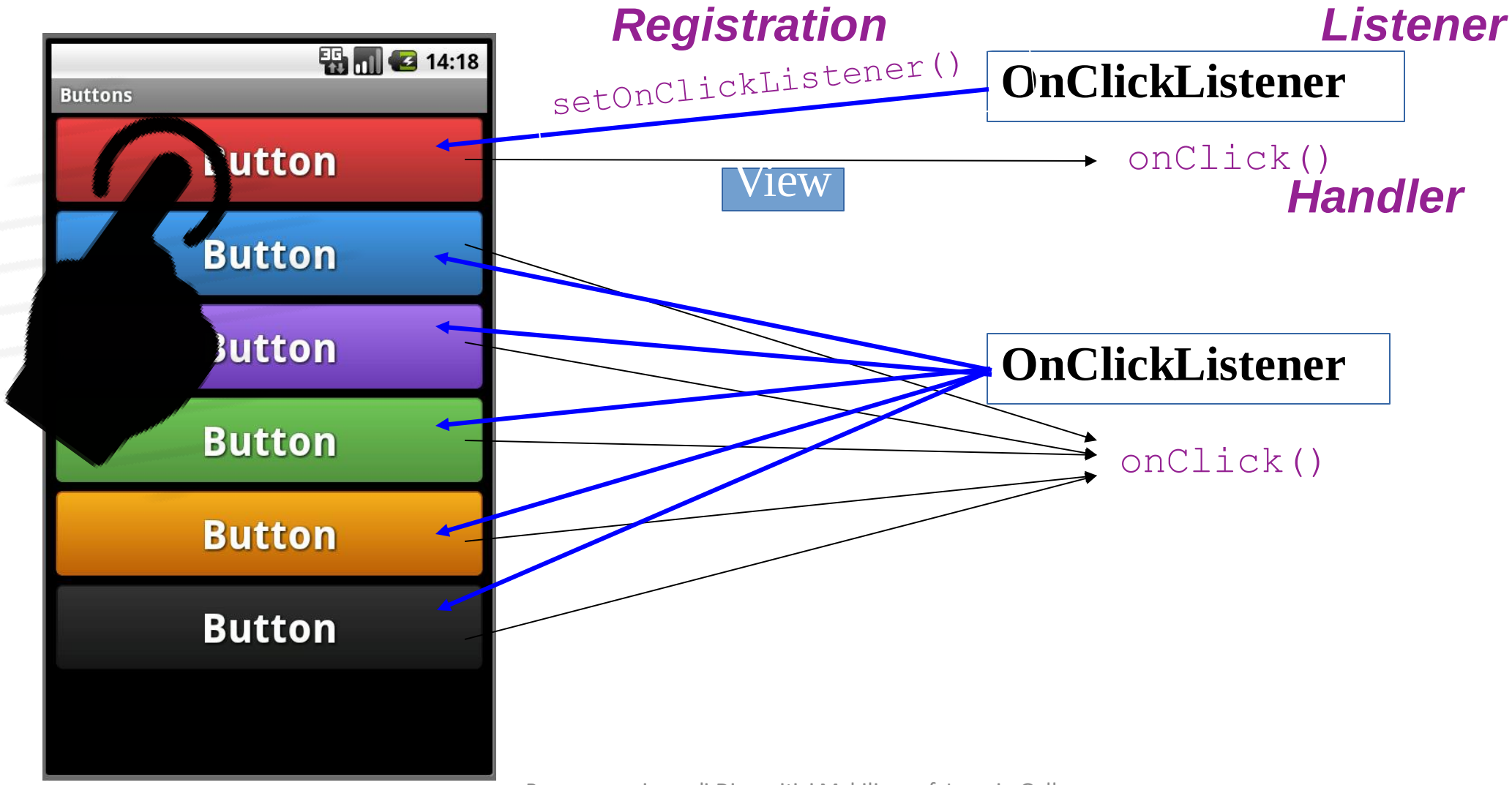
Repeat forever
- **Listener**: if an event is detected put it in the pending event list
- **Dispatcher**: take the highest priority event from the event list and send it to the appropriate **registered** event handler

An event handler ***wait*** (sleep) for an event. When sent an event by the dispatcher, it wake ups, do the prescribed list of actions and then goes back to sleep.

Common Listener Objects

Interface	Function	Description
View.OnClickListener	onClick()	This is called when the user either touches and holds the control (when in touch mode), or focuses upon the item with the navigation keys then presses the ENTER key
View.OnLongClickListener	onLongClick()	Almost the same as a click, but only longer
View.OnFocusChangeListener	onFocusChange()	When the user navigates onto or away from the control
View.OnTouchListener	onTouch()	Almost the same as click action but this handler lets you find out if the user swiped up or down. You can use this to respond to gestures
View.OnCreateContextMenuListener	onCreateContextMenu()	Android calls this when a ContextMenu is being built, as a result of a sustained long click

Android Event Management: concepts



Event Listeners & Event Handlers

Event Handler	*Event Listener & Description*	**Registering**
`onSomething()`	`OnSomethingListener`	`setOnSomethingListener()`
`onClick()`	**OnClickListener**	`setOnClickListener()`
`onLongClick()`	**OnLongClickListener**	`setOnLongClickListener()`

Event Listeners Registration

- Event Registration is the process by which an **Event Handler** gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

- There are several ways to register your event listener for any event:
  - Registering listeners
  - Implements the Listener interface
  - Defining the event handler in the <Button> element (in the XML layout file)

One Listener for Multiple events sources

- For App with many buttons you don't have to write new onClickListener for Every Button
- (1) Implement View.OnClickLister in your Activity or other class
- (2) Implement onClick() method in your Activity or class

```
class MainActivity : AppCompatActivity(), View.OnClickListener {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
  }

  override fun onClick(p0: View?) {
    TODO("Not yet implemented")
  }
}
```

```
class MyListener: View.OnClickListener{
  override fun onClick(p0: View?) {
    TODO("Not yet implemented")
  }
}
```

One Listener for Multiple events sources

- (3) Assign OnClickListener to a Button

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_main)
  button2.setOnClickListener(this)
  button3.setOnClickListener(this)
  button4.setOnClickListener(this)
  button5.setOnClickListener(this)
}
```

```
override fun onCreate(savedInstanceState: Bundle?) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.activity_main)

  val listener = MyListener()
  button2.setOnClickListener(listener)
  button3.setOnClickListener(listener)
  button4.setOnClickListener(listener)
  button5.setOnClickListener(listener)
}
```

One Listener for Multiple events sources

- (4) Find Buttons By Id and Implement Your Code

```
override fun onClick(p: View?) {
  when (p?.getId()) {
    R.id.button2 -> {print("button2")}
    R.id.button3 -> print("button3")
    R.id.button4 -> print("button4")
    R.id.button5 -> print("button5")
    else -> {
      print("unhandled event source!")
    }
  }
}
```

Handler with one param:

```
button2.setOnClickListener(object : View.OnClickListener {
  override fun onClick(v: View?) {
    toast("text message")
  }
})
```

```
fun toast(msg: String){
  Toast.makeText(this@MainActivity, msg, Toast.LENGTH_SHORT).show()
}
```

- we can use lambda

```
button2.setOnClickListener{toast("text message")}
```

- or

```
button2.setOnClickListener({ v -> toast("text message") })
```

Handeler with more than one param:

```
editText.setOnKeyListener(object : View.OnKeyListener {
    override fun onKey(v: View?, keyCode: Int, event: KeyEvent): Boolean {
      // if the event is a key down event on the enter button
      if (event.action == KeyEvent.ACTION_DOWN &&
        keyCode == KeyEvent.KEYCODE_ENTER
      ) {
        // perform action on key press
        toast("Pressed Enter Key\n\n${editText.text}")
          return true
        }
        return false
    }
  })
```

```
editText.setOnKeyListener{v, keyCode, event ->
  // if the event is a key down event on the enter button
  if (event.action == KeyEvent.ACTION_DOWN &&
    keyCode == KeyEvent.KEYCODE_ENTER
  ) {
    // perform action on key press
    toast("Pressed Enter Key\n\n${editText.text}")
    true
  }
  false
}
```

Listener: registration via XML layout

- The name of the handler method is specified in the XML definition of the element via the android:onClick attribute.
- The Event handler is defined by implementing the corresponding method in the Activity class
- The event handler method must:
  - have a void return type and take
  - a View as an argument.
- You can not handle any other event except click event using this approach.

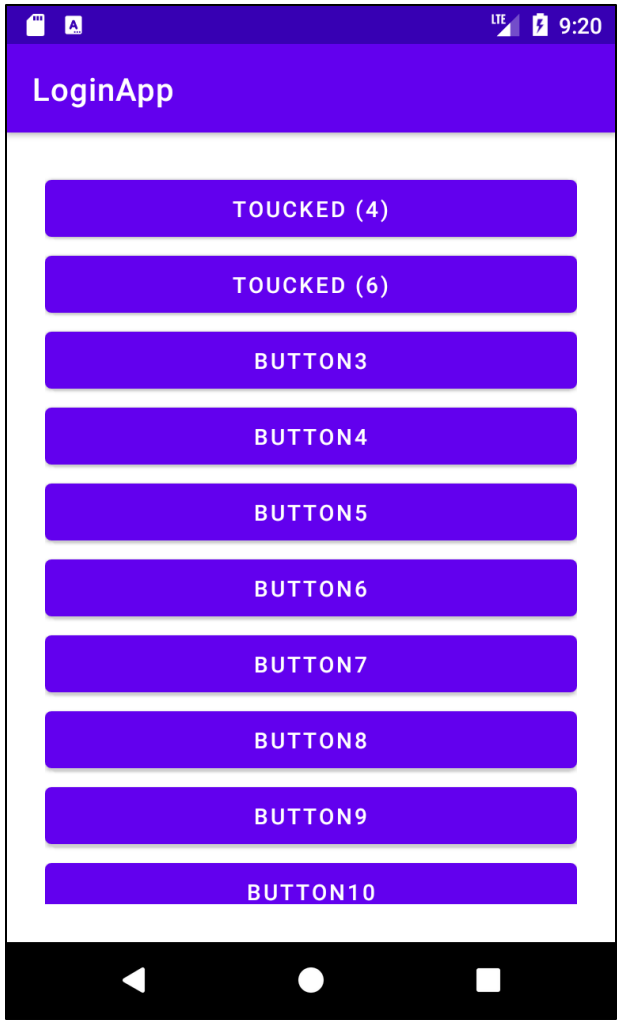
How does the android:onClick XML attribute differ from setOnClickListener?

- Those two code snippets are equal, just implemented in two different ways:

```
mybutton.setOnClickListener(object : View.OnClickListener {
  override fun onClick(v: View?) {
    toast(v)
  }
})
```


```
fun toast(v: View?){
  Toast.makeText(this, "text", 5).show()
}
```

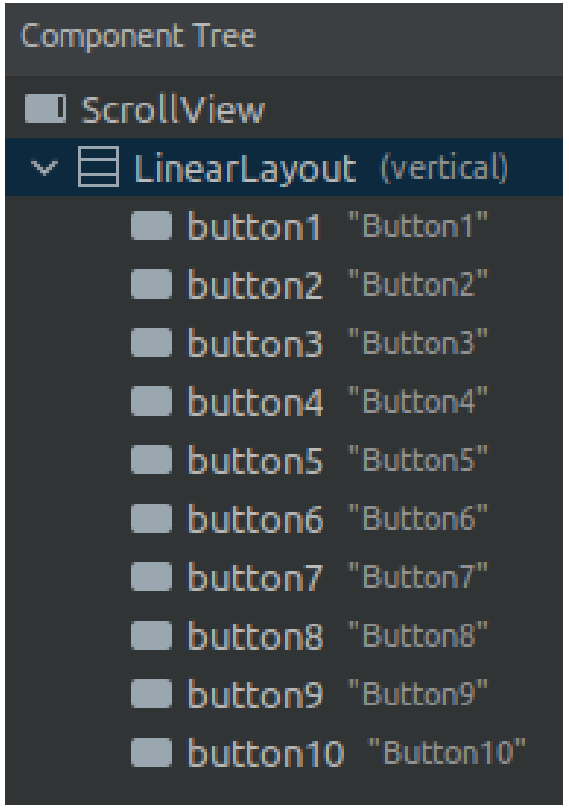
```
<?xml version="1.0" encoding="utf-8"?>
<!-- layout elements -->
<Button android:id="@+id/mybutton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Click me!"
   android:onClick="toast" />
<!-- even more layout elements -->
```

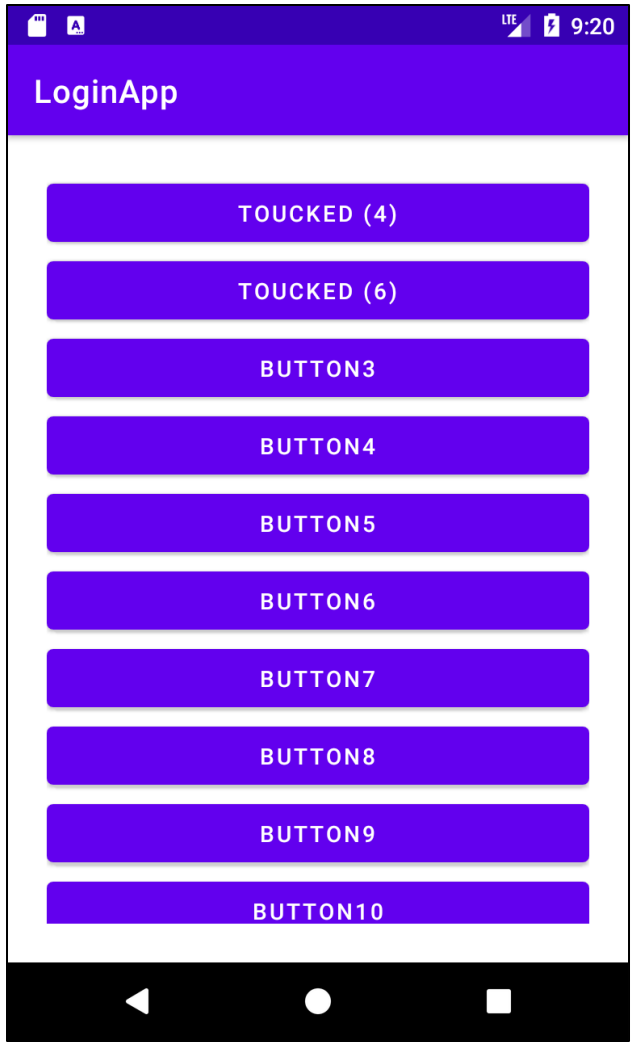


App example: EventsApp

- Each button must have a counter to be incremented by 1
with each click

Layout


Component Tree
ScrollView
LinearLayout (vertical)
button1 "Button1"
button2 "Button2"
button3 "Button3"
button4 "Button4"
button5 "Button5"
button6 "Button6"
button7 "Button7"
button8 "Button8"
button9 "Button9"
button10 "Button10"


9:20
LoginApp
TOUCKED (4)
TOUCKED (6)
BUTTON3
BUTTON4
BUTTON5
BUTTON6
BUTTON7
BUTTON8
BUTTON9
BUTTON10

Solution 1

- Using anonymous objects of type View.OnClickListener

```
class EventActivity : AppCompatActivity(){
  override fun onCreate(savedInstanceState: Bundle?){
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_event)
    button2.setOnClickListener(object : View.OnClickListener{
      var count = 0
      override fun onClick(v: View?) {
        button2.text = "toucked (${++count})"
      }
    })
    button1.setOnClickListener(object : View.OnClickListener{
      var count = 0
      override fun onClick(v: View?) {
        button1.text = "toucked (${++count})"
      }
    })
    // ...
  }
}
```

Solution 2

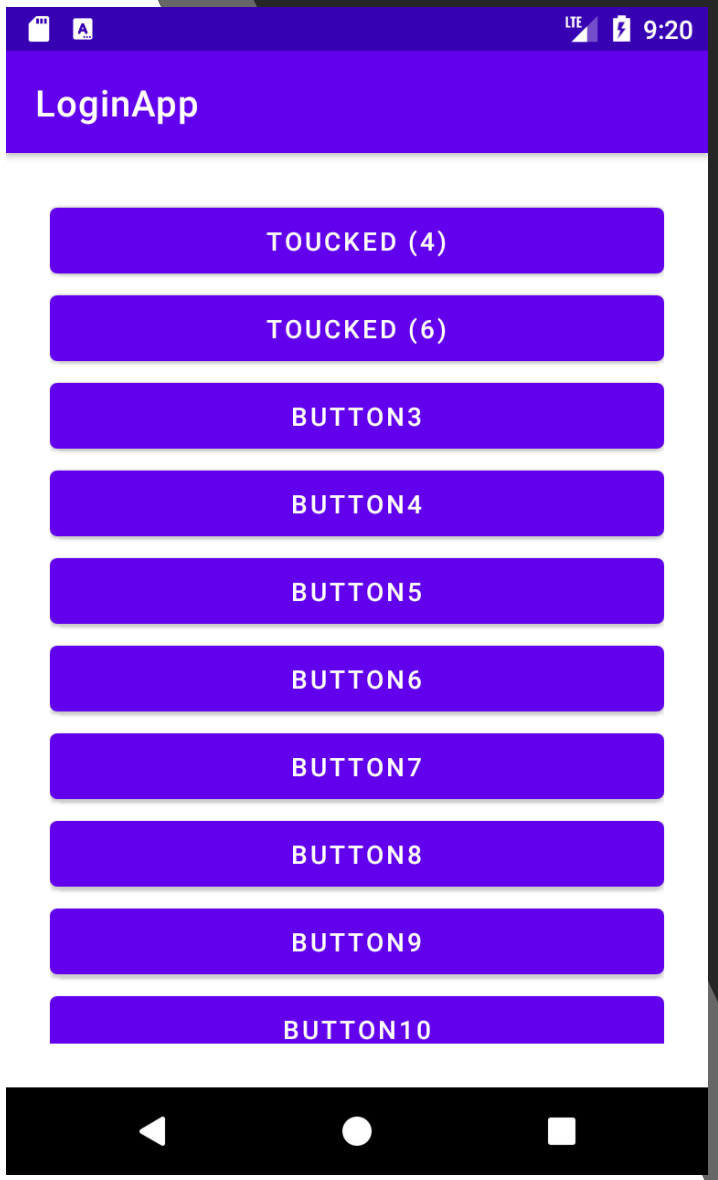
- EventActivity becomes a Listener for all Buttons

```
class EventActivity : AppCompatActivity(), View.OnClickListener {
  private var counters: MutableMap<Button?, Int?> = mutableMapOf()
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_event)
    button1.setOnClickListener(this)
    button2.setOnClickListener(this)
    button3.setOnClickListener(this)
    button4.setOnClickListener(this)
    // ...
  }
  override fun onClick(source: View?) {
    if (source is Button) {
      counters.putIfAbsent(source, 1)
      counters[source] = counters[source]!! + 1
      source.text = "toucked (${counters[source]})"
    }
  }
}
```

Solution 3

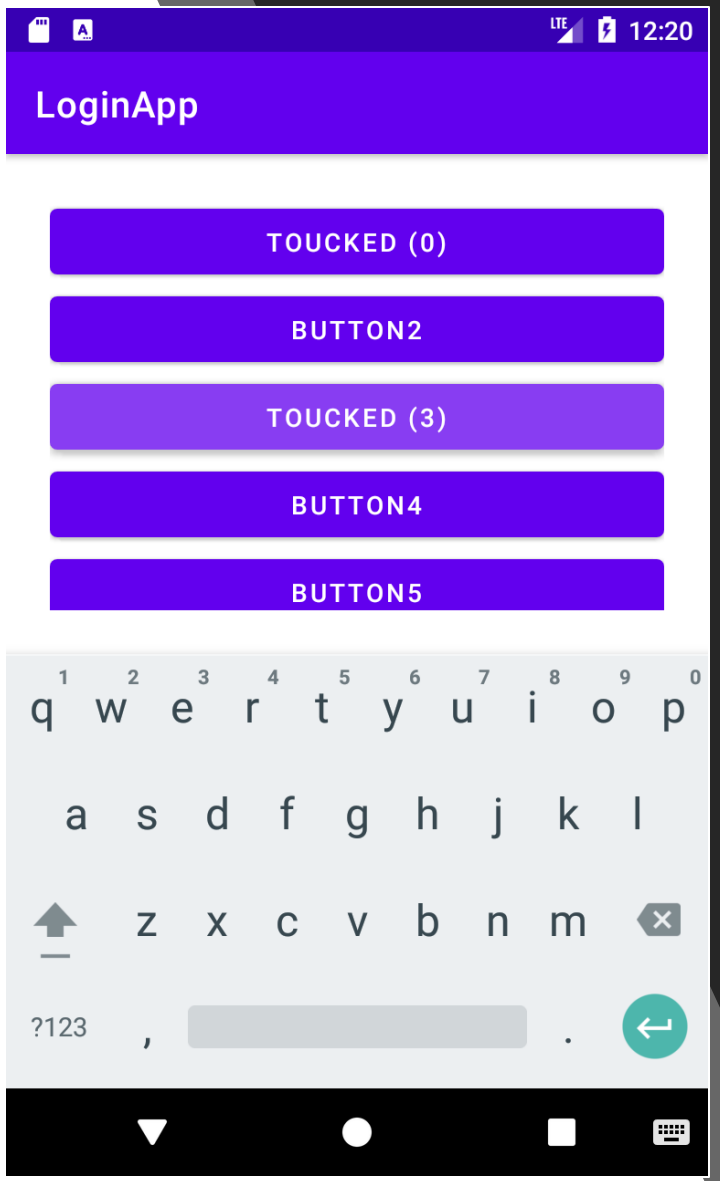
- Create a specialized Listener for all Buttons

```
class EventActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_event)
    val btnListener = ButtonListener()
    button1.setOnClickListener(btnListener)
    button2.setOnClickListener(btnListener)
    button3.setOnClickListener(btnListener)
    button4.setOnClickListener(btnListener)
    // ...
  }
}
class ButtonListener: View.OnClickListener{
  private var counters: MutableMap<Button?, Int?> = mutableMapOf()
  override fun onClick(source: View?) {
    if (source is Button) {
      counters.putIfAbsent(source, 0)
      counters[source] = counters[source]!! + 1
      source.text = "toucked (${counters[source]})"
    }
  }
}
```



Exercise 2

Each button
must have a counter to be incremented by 1
with each click
and
by 10 with each long click



Exercise 3

The counter of each Button
must be put to ZERO
using the DEL key of the keyboard

Some changes

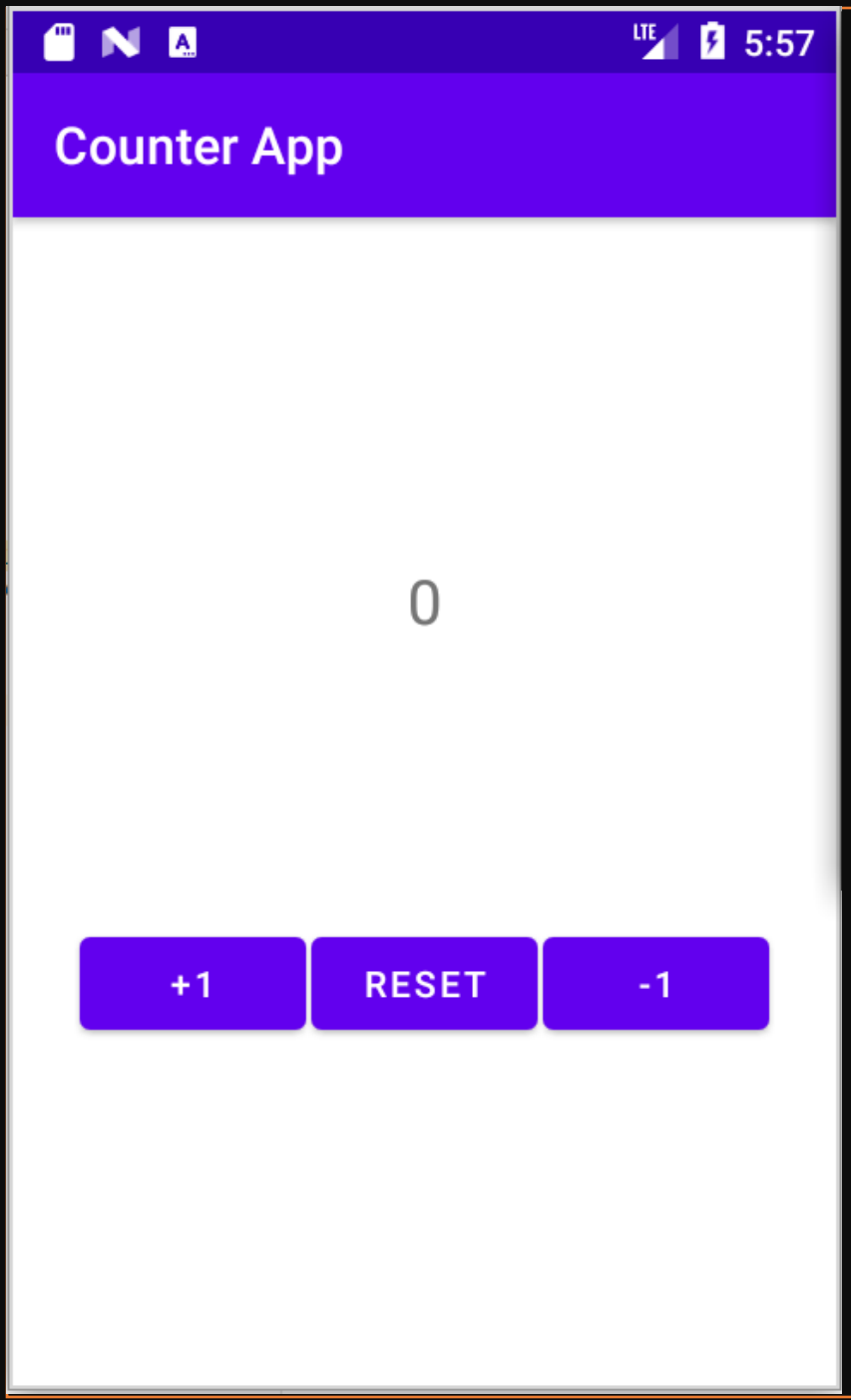
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_event)

    val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
    val btnClickListener = ButtonListenerClick(counters, imm)
```

- InputMethodManager is needed to show the keyboard

```
<Button
    android:id="@+id/button1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Button1"
    android:focusableInTouchMode="true"/>
```

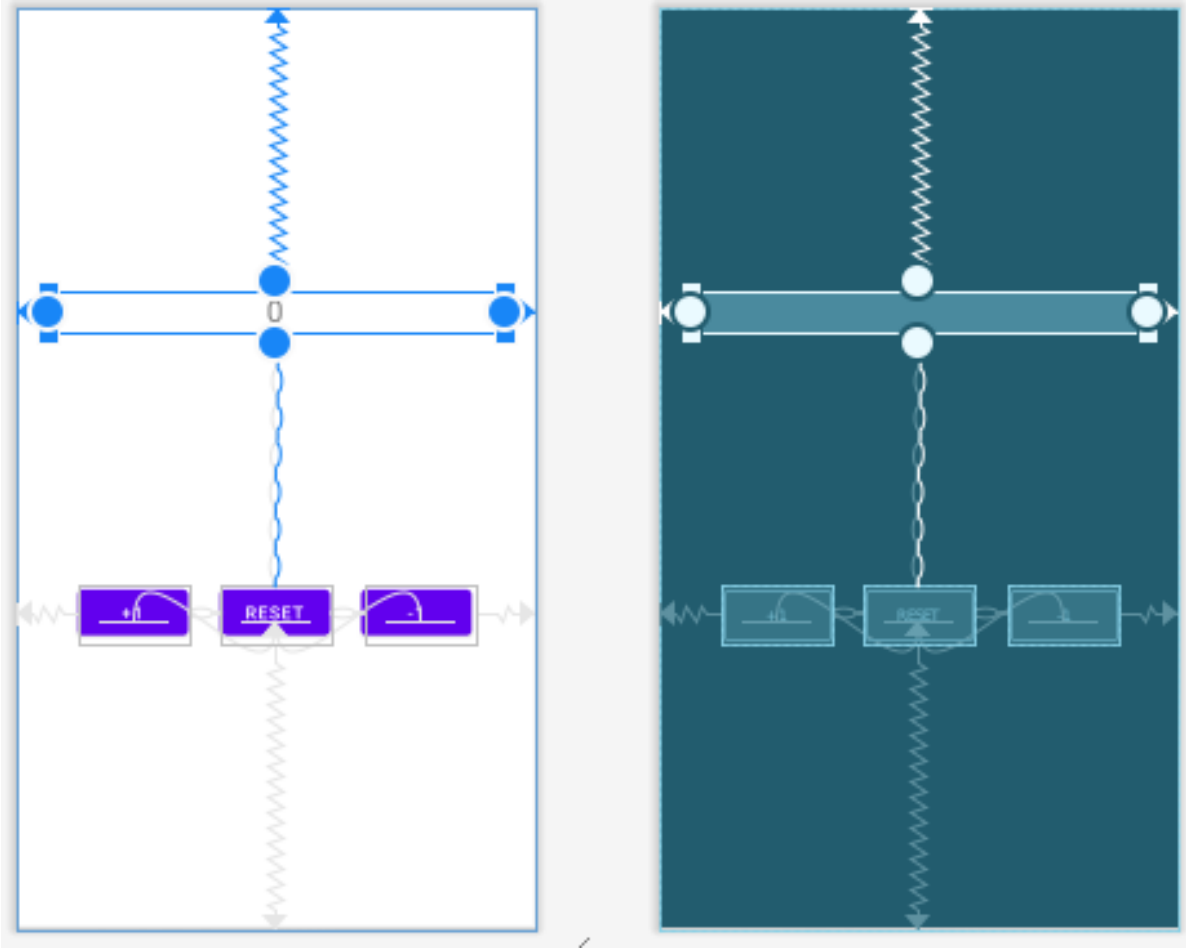
- Button must be focusable

```
class ButtonListenerClick(var counters: MutableMap<Button?, Int?>,
                          val imm: InputMethodManager): View.OnClickListener{
    override fun onClick(source: View?) {
        if (source is Button) {
            counters.putIfAbsent(source, 0)
            counters[source] = counters[source]!! + 1
            source.text = "toucked (${counters[source]})"

            imm.showSoftInput(source, flags: 0)
        }
    }
}
```



App example: CounterApp

Layout



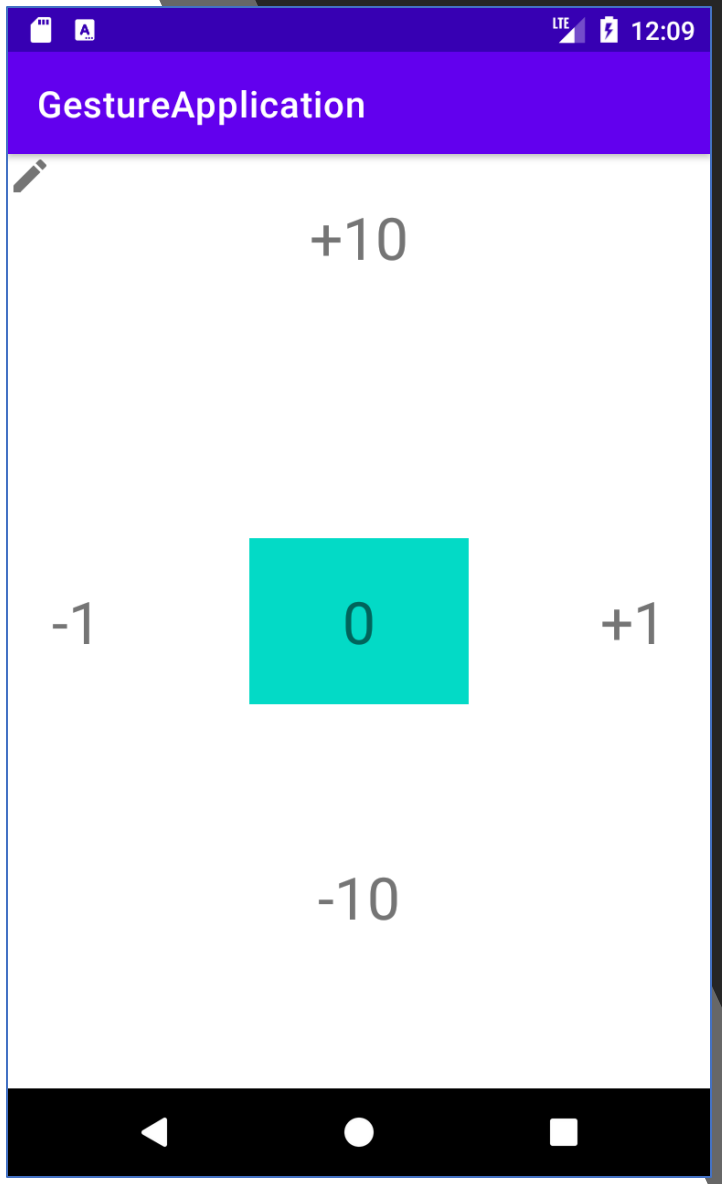
Layout

```
<?xml version="1.0"  encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="24dp"
tools:context=".MainActivity">

<TextView
android:id="@+id/textView"
android:layout_width="0dp"
android:layout_height="wrap_content"
android:text="0"
android:textSize="24sp"
android:gravity="center"
app:layout_constraintBottom_toTopOf="@+id/buttonReset"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"  />
```

```
<Button
android:id="@+id/buttonPlus"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="+1"
app:layout_constraintBaseline_toBaselineOf="@+id/buttonReset"
app:layout_constraintEnd_toStartOf="@+id/buttonReset"
app:layout_constraintHorizontal_chainStyle="spread"
app:layout_constraintStart_toStartOf="parent" />
<Button
android:id="@+id/buttonReset"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Reset"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toStartOf="@+id/buttonMinus"
app:layout_constraintStart_toEndOf="@+id/buttonPlus"
app:layout_constraintTop_toBottomOf="@+id/textView" />
<Button
android:id="@+id/buttonMinus"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="-1"
app:layout_constraintBaseline_toBaselineOf="@+id/buttonReset"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toEndOf="@+id/buttonReset" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Kotlin code

```
class MainActivity : AppCompatActivity(), View.OnClickListener {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    buttonPlus.setOnClickListener(this)
    buttonReset.setOnClickListener(this)
    buttonMinus.setOnClickListener(this)
  }

  override fun onClick(v: View?) {
    when(v?.id){
      R.id.buttonPlus -> textView.text = "${textView.text.toString().toInt() + 1}"
      R.id.buttonMinus -> textView.text = "${textView.text.toString().toInt() - 1}"
      R.id.buttonReset -> textView.text = "0"
    }
  }
}
```



Gesture events

- OnTouch
- GestureDetection
- OnDrag

Detecting gestures

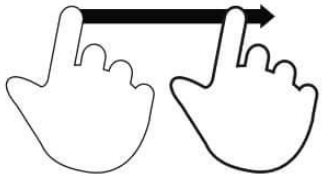
Tap

Briefly touch surface with fingertip

Double tap

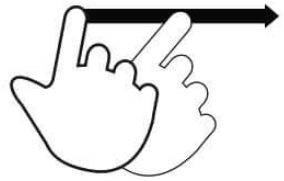
Rapidly touch surface twice with fingertip

Drag



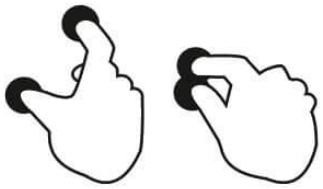
Move fingertip over surface without losing contact

Flick



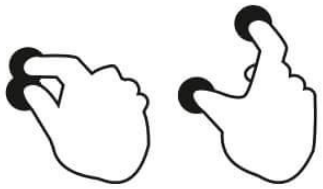
Quickly brush surface with fingertip

Pinch



Touch surface with two fingers and bring them closer together

Spread



Touch surface with two fingers and move them apart

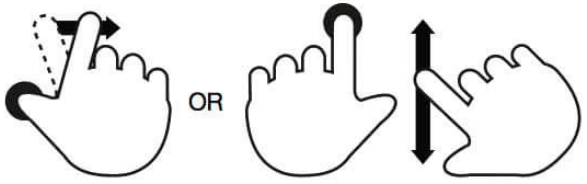
Press

Touch surface for extended period of time

Press and tap

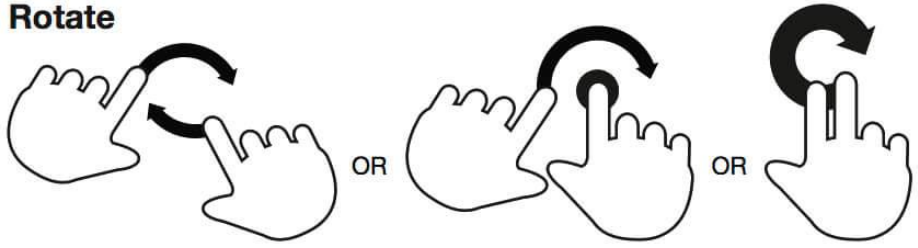
Press surface with one finger and briefly touch surface with second finger

Press and drag



Press surface with one finger and move second finger over surface without losing contact

Rotate



Touch surface with two fingers and move them in a clockwise or counterclockwise direction

```
override fun onTouch(view: View?, motion: MotionEvent?): Boolean {
    when (motion?.action) {
        MotionEvent.ACTION_DOWN -> {
            xStart = motion?.x
            yStart = motion?.y
            return true
        }
        MotionEvent.ACTION_MOVE -> {...}
        MotionEvent.ACTION_UP -> {
            var xDiff = motion?.x - xStart
            var yDiff = motion?.y - yStart
            if (abs(xDiff) > abs(yDiff)){ // horizontal movement
                if(xDiff>0) ++counter else --counter
            }else {  // vertical movement
                counter = if(xDiff>0) counter-10 else counter+10
            }
            tv_display.text = "$counter"
            return true
        }
        else -> {
            return false
        }
    }
}
```

```
class MainActivity : AppCompatActivity(), View.OnTouchListener {
    var xStart = 0.0F
    var yStart = 0.0F
    var counter = 0
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        cl.setOnTouchListener(this)
    }
    override fun onTouch(view: View?, motion: MotionEvent?): Boolean {...}
}
```

OnTouch

```
MotionEvent.ACTION_MOVE -> {
    ivPencil.x = motion?.x
    ivPencil.y = motion?.y
    return true
}
```

Better Swipe events

- Problems:
  - even a single click/touch is interpreted as a swipe
  - even a very slow movement is interpreted as a swipe
- Solutions:
  - add a minimum shift as a threshold
  - add a minimum speed of movement

```
MotionEvent.ACTION_UP -> {
    val elapsedSec = (System.nanoTime()-begin)*1e-9
    var xDiff = motion?.x - xStart
    var yDiff = motion?.y - yStart
    if (abs(xDiff) > abs(yDiff)){ // horizontal movement
        if (abs(xDiff) < SWIPE_THRESHOLD)
            return false
        if (abs(xDiff)/elapsedSec < SWIPE_VELOCITY_THRESHOLD)
            return false
        if (xDiff>0) ++counter else --counter
    }else {  // vertical movement
        if (abs(yDiff) < SWIPE_THRESHOLD)
            return false
        if (abs(yDiff)/elapsedSec < SWIPE_VELOCITY_THRESHOLD)
            return false
        counter = if(xDiff>0) counter-10 else counter+10
    }
    tv_display.text = "$counter"
    return true
}
```

```
class MainActivity : AppCompatActivity(), View.OnTouchListener{
    private val SWIPE_THRESHOLD: Int = 100 // In pixel
    private val SWIPE_VELOCITY_THRESHOLD: Int = 200 // In pixel/sec
```

```
override fun onTouch(view: View?, motion: MotionEvent?): Boolean {
    when (motion?.action) {
        MotionEvent.ACTION_DOWN -> {
            begin = System.nanoTime()
```

GestureDetector

- Detects various gestures and events using the supplied MotionEvents.
- Nested classes:
  - SimpleOnGestureListener
    A convenience class to extend when you only want to listen for a subset of all the gestures.
  - OnGestureListener
    The listener that is used to notify when gestures occur.
  - ...

GestureDetector

```
private inner class GestureListener : GestureDetector.SimpleOnGestureListener() {
    private val SWIPE_THRESHOLD = 100
    private val SWIPE_VELOCITY_THRESHOLD = 100
    override fun onFling(e1: MotionEvent, e2: MotionEvent, velocityX: Float, velocityY: Float): Boolean {
        try {
            val diffY = e2.y - e1.y
            val diffX = e2.x - e1.x
            if (Math.abs(diffX) > Math.abs(diffY)) {
                if (Math.abs(diffX) > SWIPE_THRESHOLD && Math.abs(velocityX) > SWIPE_VELOCITY_THRESHOLD) {
                    if (diffX > 0) counter++ else  counter--
                }
            } else {
                if (Math.abs(diffY) > SWIPE_THRESHOLD && Math.abs(velocityY) > SWIPE_VELOCITY_THRESHOLD) {
                    if (diffY > 0) counter-=10 else  counter+=10
                }
            }
            tv_display.text = "$counter"
        } catch (exception: Exception) {
            exception.printStackTrace()
        }
        return true
    }
}
```

```
class MainActivity : AppCompatActivity(), View.OnTouchListener{
    var counter = 0
    lateinit var detector: GestureDetector

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        detector = GestureDetector( context: this, GestureListener())
        cl.setOnTouchListener(this)
    }

    override fun onTouch(view: View?, motion: MotionEvent?): Boolean {
        detector.onTouchEvent(motion)
        return true
    }

    private inner class GestureListener : GestureDetector.SimpleOnGestureListener() {...}
}
```