Advanced techniques:

Touch and Swipes

Dr. Abraham (Abey) Campbell

Learning Objectives

- Detect and handle touches, swipes, taps, and gestures

Touches and Gestures (1 of 2)

- Commonly used in apps
- We first explore touches, swipes, and taps.
- Then we build a puzzle app.

Touches and Gestures (2 of 2)

- Same concept as general event handling
- Implement an interface that the event
- Create an object of that class, a listener
- Register that object on some GUI component(s)

Puzzle


5554:Nexus_5_API_23_x86
5:17
PuzzleV2
USING
JAVA
MOBILE
I LOVE
PROGRAMMING

Touch Event (1 of 2)

- We first build a practice Touches app (Version 0).
- We are still not building the puzzle app yet.

Touch Event (2 of 2)

- View.OnTouchListener is a public static inner interface of View.
- It includes the onTouch method (“callback” method).
- onTouch is called when a touch event happens on a View that a View.OnTouchListener is registered on.

onTouch Method

boolean onTouch(View v, MotionEvent me)

- v = View that originated the event
- me = contains info about the touch event

Touch Event

- We implement the onTouch method.
- If we return true ➔ the event is consumed.
- If we return false ➔ the event is propagated to Views underneath the current view.

Implement View.OnTouchListener

```
// private class inside the MainActivity class
private class TouchHandler implements
            View.OnTouchListener {
  // the TouchHandler class needs to override onTouch
  public boolean onTouch( View v, MotionEvent event ) {
      // process the event here
  }
}
```

Registering a Touch Event

- The setOnTouchListener method (of the View class) is used to register a Touch listener on a GUI component:

```
void setOnTouchListener (
    View.OnTouchListener listener )
```

Touch Event (1 of 4)

- If we want the View v to respond to touch events.
- Assume that TouchHandler implements View.OnTouchListener.

```
TouchHandler th = new TouchHandler( );
v.setOnTouchListener( th );
```

Touch Event (2 of 4)

- Instead of coding a separate class implementing View.OnTouchListener, we can have the current Activity class implement View.OnTouchListener.
- ➔ “this” is a View.OnTouchListener.

Touch Event (3 of 4)

```
public class MainActivity extends
        AppCompatActivity implements
        View.OnTouchListener {
    // code MainActivity here
    // implement onTouch method
}
```

Touch Event (4 of 4)

- To register the listener (this Activity object) on a View component named v:

```
v.setOnTouchListener( this );
```

- The onClick method will be called when a touch event occurs; one of its parameter is a MotionEvent.

MotionEvent Class (1 of 2)

Method	Description
float getRawX( )	Returns x coordinate of the touch within the screen
float getRawY( )	Returns y coordinate of the touch within the screen
float getX( )	Returns x coordinate of the touch within the View that originated the event
float getY( )	Returns x coordinate of the touch within the View that originated the event

MotionEvent Class (2 of 2)

Method	Description
int getAction( )	Returns the type of action that occurred within the touch event; the return value can be compared to one of the constants below.

Constant	Description
ACTION_DOWN	The user touched the screen
ACTION_UP	The user stopped touching the screen
ACTION_MOVE	The user is moving his or her finger on the screen

Touch Event

- We can retrieve the type of action with the MotionEvent parameter:

```
int action = me.getAction( );
```

Implement onTouch (1 of 2)

```
public boolean onTouch( View v,
                        MotionEvent event ) {
    // retrieve the action
    int action = event.getAction( );
    // do something depending on the action
}
```

Implement onTouch (2 of 2)

```
switch( action ) {
  case MotionEvent.ACTION_DOWN:
    Log.w( MA, "DOWN: v = " + v + "; event = " + event );
    break;
  case MotionEvent.ACTION_MOVE:
    Log.w( MA, "MOVE: v = " + v + "; event = " + event );
    break;
  case MotionEvent.ACTION_UP:
    Log.w( MA, "UP: v = " + v + "; event = " + event );
    break;
}
```

Touch Event

- If we run the app, touch the screen, swipe, and lift up her finger, we get the following:
  - One ACTION_DOWN event
  - Many ACTION_MOVE events
  - One ACTION_UP event

Output when Swiping the Screen

view = android.widget.FrameLayout{92f6592 V.E..... .....I. 0,0-0,0 #1020002 android:id/content}

DOWN: v = android.widget.FrameLayout{92f6592 V.E..... ..... 0,72-1080,1776 #1020002 android:id/content}; event = MotionEvent { action=ACTION_DOWN, actionButton=0, id[0]=0, x[0]=336.0, y[0]=1130.0, toolType[0]=TOOL_TYPE_FINGER, buttonState=0, metaState=0, flags=0x0, edgeFlags=0x0, pointerCount=1, historySize=0, eventTime=66765, downTime=66765, deviceId=0, source=0x1002 }

MOVE: v = android.widget.FrameLayout{92f6592 V.E..... ..... 0,72-1080,1776 #1020002 android:id/content}; event = MotionEvent { action=ACTION_MOVE, actionButton=0, id[0]=0, x[0]=482.5914, y[0]=1061.9808, toolType[0]=TOOL_TYPE_FINGER, …

Touch Event (1 of 16)

- The various MOVE actions form a discrete (not continuous) set of actions representing the swipe.

Touch Event (2 of 16)

- Next, we place a label on the screen and move it with our finger.
- We are still not building the puzzle app yet.
- We continue to build our practice Touches app, Version 1.

Touch Event (3 of 16)

- MainActivity implements View.OnTouchListener.
- We override onTouch inside MainActivity.
- “this” is a View.OnTouchListener.

Touch Event (4 of 16)

- To move the label, we need:
  - The original position of the label
  - The touch position
- Inside onTouch, we retrieve the current touch position and move the label accordingly.

Touch Event (5 of 16)

- We need to access the original label position and the touch position when the DOWN action happens.
- We need to use them to compute the new label position when the MOVE action happens.

Touch Event (6 of 16)

- Thus, we declare four instance variables: startX, startY, startTouchX, and startTouchY.
- To move the label, we need to change its x and y coordinates.

Touch Event (7 of 16)

- To change its x and y coordinates, we change its layout parameters using the setLayoutParams method.
- The setLayoutParams method of the View class enables us to change the layout parameters of a View (the label here).

Touch Event (8 of 16)

void setLayoutParams(

ViewGroup.LayoutParams params)

- We place or label inside a RelativeLayout (there is an AbsoluteLayout class but it is now deprecated).

RelativeLayout rl = new RelativeLayout(this);

Touch Event (9 of 16)

- RelativeLayout.LayoutParams inherits from ViewGroup.LayoutParams.
- ➔ a RelativeLayout.LayoutParams “is a” ViewGroup.LayoutParams.

Touch Event (10 of 16)

- To create a RelativeLayout.LayoutParams for a label, we use a constructor with two parameters, representing the width and height of the View (our label).

RelativeLayout.LayoutParams(int w, int h)

Touch Event (11 of 16)

- Since we use the RelativeLayout.LayoutParams at several places (onCreate for the original position, and onTouch when we move the label), we declare it as an instance variable:

```
private RelativeLayout.LayoutParams
params;
```

Touch Event (12 of 16)

- We initialize params inside onCreate:

```
params = new
RelativeLayout.LayoutParams( 300, 200 );
```

- We need to specify the x and y coordinates of params before we assign it to our label.

Touch Event (13 of 16)

- RelativeLayout.LayoutParams inherits from ViewGroup.MarginLayoutParams.
- It includes the public fields bottomMargin, topMargin, leftMargin, and rightMargin.

```
params.leftMargin = 50;
params.topMargin = 150;
```

Touch Event (14 of 16)

- Now we can add the label to the RelativeLayout:

  rl.addView(tv, params);

- Set the content view for the activity:

  setContentView(rl);

Touch Event (15 of 16)

- The GUI is only made up of one label; we create it programmatically

```
// tv is a TextView instance variable
tv = new TextView( this );
tv.setBackgroundColor( 0xFFFF0000 );
```

Touch Event (16 of 16)

- We add touch listener to our label:

```
tv.setOnTouchListener( this );
```

onTouch Method

- We only care about the DOWN action and the MOVE action.
- DOWN action ➔ get the starting x and y coordinates of the label and of the touch.

DOWN Action (1 of 2)

- Get the x and y starting coordinates for the label:

```
startX = params.leftMargin;
startY = params.topMargin;
```

DOWN Action (2 of 2)

- Get x and y starting coordinates for the touch:

```
startTouchX = ( int ) event.getX( );
startTouchY = ( int ) event.getY( );
```

MOVE Action (1 of 3)

- Calculate the new x coordinate of the label.
- New x coordinate = old x coordinate + x coordinate of current touch – x coordinate of original touch.
- params.leftMargin is the x coordinate of the label.

MOVE Action (2 of 3)

params.leftMargin = startX + (int)
event.getX( ) – startTouchX;

- Same for y coordinate:

params.topMargin = startY + (int)
event.getY( ) – startTouchY;

MOVE Action (3 of 3)

- Run the app.
- If we touch the label and move our finger, the label moves along.
- If we touch outside the label and move our finger, nothing happens (because the listener—“this”—is registered on the label only).

Puzzle App—Version 0

- We code a very simple Model class for the puzzle (we want to focus on touches, not model complexity).
- The puzzle is hard coded:
  - I LOVE – MOBILE – PROGRAMMING - USING - JAVA

Puzzle Model

- We include a default constructor, a solved method (to check that the parts of the puzzle are in the correct order), a scramble method (to randomly order the parts), and an accessor for the number of parts.
- See Puzzle.java (next slide).

```java
public class Puzzle {
  public static final int NUMBER_PARTS = 5;
  String [] parts;
  Random random = new Random( );

  public Puzzle( ) {
    parts = new String[NUMBER_PARTS];
    parts[0] = "I LOVE";
    parts[1] = "MOBILE";
    parts[2] = "PROGRAMMING";
    parts[3] = "USING";
    parts[4] = "JAVA";
  }

  public boolean solved( String [] solution ) {
    if( solution != null && solution.length == parts.length ) {
      for( int i = 0; i < parts.length; i++ ) {
        if( !solution[i].equals( parts[i] ) )
          return false;
      }
      return true;
    }
    else
      return false;
  }

  public String [] scramble( ) {
    String [] scrambled = new String[parts.length];
    for( int i = 0; i < scrambled.length; i++ )
      scrambled[i] = parts[i];

    while( solved( scrambled ) ) {
      for( int i = 0; i < scrambled.length; i++ ) {
        int n = random.nextInt( scrambled.length - i ) + i;
        String temp = scrambled[i];
        scrambled[i] = scrambled[n];
        scrambled[n] = temp;
      }
    }
    return scrambled;
  }

  public int getNumberOfParts( ) {
    return parts.length;
  }
}
```

Puzzle App (1 of 2)

- We update the manifest so that the app only runs in vertical orientation

  android:screenOrientation="portrait"

- We specify a text size to 32sp in styles.xml

<item name="android:textSize">32sp</item>

Puzzle App (2 of 2)

- We code a separate View for the puzzle, PuzzleView.java.
- MainActivity.java is the app's Controller; it sets the View to a PuzzleView and scrambles the pieces of the puzzle.

Puzzle View

- Inside PuzzleView, we have three instance variables.
  - An array of TextViews
  - An array of LayoutParams that parallels the TextViews
  - An array of colors (randomly generated each time the app runs) that also parallels the TextViews

PuzzleView Constructor

- The PuzzleView constructor takes four parameters:
  - an Activity (we need it to instantiate the TextViews)
  - an int for the width of this View
  - an int for the height of this View, and
  - another int, the number of pieces in the puzzle (equal to the number of TextViews)

Puzzle View (1 of 3)

- The PuzzleView constructor calls the buildGuiByCode method, passing all four parameters.
- The buildGuiByCode method builds the GUI.

Puzzle View (2 of 3)

```
public PuzzleView( Activity activity, int
   width, int height, int numberOfPieces ) {
     super( activity );
     buildGuiByCode( activity, width, height,
                                  numberOfPieces );
}
```

Puzzle View (3 of 3)

- Each TextView is as wide as the screen.
- The height of each TextView is the height of screen divided by the number of pieces in the puzzle.

Building the Puzzle GUI

- We instantiate the three arrays.
- We calculate the height of the TextViews (height / number of pieces).
- We create the colors randomly.
- We instantiate the TextViews.
- We size them and position them.
- We add them to this PuzzleView.

buildGuiByCode Method

```
public void buildGuiByCode( Activity activity,
        int width, int height, int numberOfPieces ) {
    tvs = new TextView[numberOfPieces];
    colors = new int[tvs.length];
    params =
      new RelativeLayout.LayoutParams[tvs.length];
    labelHeight = height / numberOfPieces;
    …
}
```

Puzzle View (1 of 5)

```
for( int i = 0; i < tvs.length; i++ ) {
  // instantiate tvs[i]
  // color background of tvs[i]
  // instantiate params[i]
  // add tvs[i] using params[i] to PuzzleView
}
```

Puzzle View (2 of 5)

- To generate a random color, we generate three random integers between 0 and 255 and use the static rgb method of the Color class (it takes three int parameters, representing the red, green, and blue components).

Puzzle View (3 of 5)

```
// inside the loop; positioning the TextViews
params[i] = new LayoutParams( width,
        labelHeight );
params[i].leftMargin = 0;
params[i].topMargin = labelHeight * i;
```

Puzzle View (4 of 5)

- Last statement of the loop: we add the current TextView using the current layout parameters:

```
addView( tvs[i], params[i] );
```

Puzzle View (5 of 5)

- We provide a method to fill the TextViews with text (which will be provided by the Model):

```
public void fillGui( String [] scrambledText ) {
    for( int i = 0; i < tvs.length; i++ )
      tvs[i].setText( scrambledText[i] );
}
```

Activity Class (1 of 3)

- In the Activity class (the Controller), we have two instance variables: a Puzzle (the Model for the app) and a PuzzleView (the View for the app).
- The onCreate method instantiates both, asks the Model for an array of scrambled puzzle text and fills the PuzzleView with it.

Activity Class (2 of 3)

- When we instantiate the PuzzleView, we need to pass the available dimensions (width and height) to the PuzzleView constructor.

Activity Class (3 of 3)

```
Point size = new Point( );
getWindowManager( ).getDefaultDisplay( )
                                   .getSize( size );
int screenHeight = size.y;
int puzzleWidth = size.x;
```

- The problem is that screenHeight includes the height of the action and status bars.

Action Bar and Status Bar Heights

- Current height of action bar = 56dp
- Current height of status bar = 24dp
- They can be retrieved dynamically (in case they change in future versions).

Retrieve the Action Bar Height

- We first assign a default value to the action bar height (56 dp).
- Then we try to retrieve that value dynamically.

Set the Default Height of the Action Bar

```
Resources res = getResources( );
DisplayMetrics metrics =
    res.getDisplayMetrics( );
float pixelDensity = metrics.density;
int actionBarHeight = ( int ) ( pixelDensity *
    ACTION_BAR_HEIGHT );
```

Retrieve the Action Bar Height

```
TypedValue tv = new TypedValue( );
if( getTheme( ).resolveAttribute(
     android.R.attr.actionBarSize, tv, true ) )
actionBarHeight =
 TypedValue.complexToDimensionPixelSize(
 tv.data, metrics );
```

Retrieve the Status Bar Height (1 of 2)

- We first assign a default value to the status bar height (24 dp):

```
int statusBarHeight = ( int ) ( pixelDensity *
    STATUS_BAR_HEIGHT );
```

- Then, we try to retrieve the value dynamically.

Retrieve the Status Bar Height (2 of 2)

```
int resourceId =
    res.getIdentifier( "status_bar_height",
    "dimen", "android" );
if( resourceId != 0 ) // found resource
   statusBarHeight =
      res.getDimensionPixelSize( resourceId );
```

Activity Class (1 of 2)

- We retrieve the height of the action and status bars and subtract them from the height of the screen.

```
int puzzleHeight = screenHeight -
    statusBarHeight - actionBarHeight;
puzzleView = new PuzzleView( this,
    puzzleWidth, puzzleHeight,
    puzzle.getNumberOfParts( ) );
```

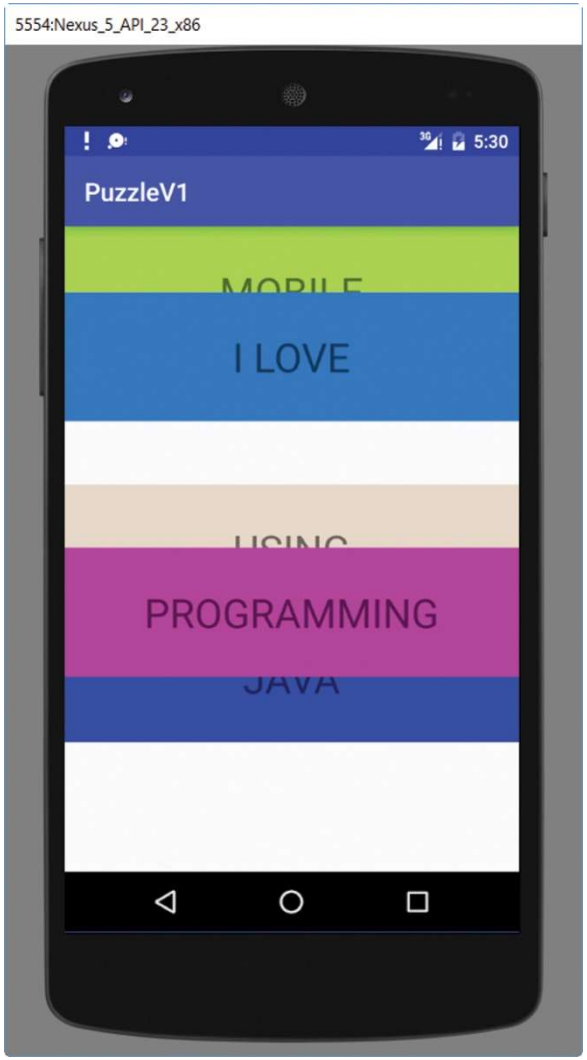
Activity Class (2 of 2)

- Once puzzle and puzzleView are instantiated, we scramble the puzzle, fill puzzleView, and set the puzzleView as the content view of this activity.

```
String [ ] scrambled = puzzle.scramble( );
puzzleView.fillGui( scrambled );
setContentView( puzzleView );
```

Puzzle Version 1 (1 of 5)

- We now have a puzzle that we can display, but we cannot move its pieces yet.
- In Version 1, we enable the user to move the pieces (without placing them in the right slots).



Puzzle Version 1 (2 of 5)

- We need to set up event handling so that we enable the user to move the pieces.
- We do that in the Controller, the MainActivity class.

Puzzle Version 1 (3 of 5)

- MainActivity implements View.OnTouchListener

```
public class MainActivity extends
    AppCompatActivity implements
                        View.OnTouchListener
```

Puzzle Version 1 (4 of 5)

- We register all the TextViews on this (mainActivity “is a” View.OnTouchListener).

```
puzzleView.enableListener( this );
```

- ➔ We need to add the enableListener method in the PuzzleView class. It registers a View.OnTouchListener on all its TextViews.

Puzzle Version 1 (5 of 5)

- We implement the onTouch method inside MainActivity:

```
public boolean onTouch( View v,
  MotionEvent event ) {
   ..
}
```

onTouch Method (1 of 4)

- We retrieve the index of the TextView being moved (➔ we need a method for this in the PuzzleView class).
- DOWN action: we capture the starting y positions (of the TexView and the touch) and bring the current TextView to the front (➔ we need methods for this in PuzzleView).
- MOVE action: we move the TextView (➔ we need a method for this in PuzzleView).

onTouch Method (2 of 4)

- First, we retrieve the index of the TextView being moved.

```
int index = puzzleView.indexOfTextView( v );
int action = event.getAction( );
```

- We need to code the indexOfTextView method in the PuzzleView class.

onTouch Method (3 of 4)

- DOWN action: We capture the starting y positions and bring the current TextView to the front.

```
case MotionEvent.ACTION_DOWN:
  puzzleView.updateStartPositions( index, (
    int ) event.getY( ) );
puzzleView.bringChildToFront( v );
```

- We need to code the updateStartPositions method in PuzzleView. The bringChildToFront method is an exiting method of View.

onTouch Method (4 of 4)

- MOVE action: move the TextView.

```
case MotionEvent.ACTION_MOVE:
   puzzleView.moveTextViewVertically(
         index, ( int ) event.getY( ) );
```

- We need to code the moveTextViewVertically method in PuzzleView.

PuzzleView Class (1 of 6)

- We need to add the methods in the PuzzleView class that we call in the onTouch method of the MainActivity class:

```
public int indexOfTextView( View tv )
public void updateStartPositions( int index, int y )
public void moveTextViewVertically( int index, int y )
public void enableListener( View.OnTouchListener listener )
```

PuzzleView Class (2 of 6)

- We also need to add two instance variables to store the starting y coordinates of the touch and the TextView being moved.
- These are used in two methods (updateStartPositions and moveTextViewVertically), so they cannot be local variables.

PuzzleView Class (3 of 6)

- The enableListener method registers all the TextViews on a listener (this MainActivity when the method is called):

```
public void enableListener( View.OnTouchListener listener ) {
   for( int i = 0; i < tvs.length; i++ )
      tvs[i].setOnTouchListener( listener );
}
```

PuzzleView Class (4 of 6)

- The indexOfTextView returns the index of a TextView within the arrays tvs:

```
public int indexOfTextView( View tv ) {
  if( ! ( tv instanceof TextView ) )
      return -1;
  for( int i = 0; i < tvs.length; i++ ) {
      if( tv == tvs[i] )
        return i;
  }
  return -1;
}
```

PuzzleView Class (5 of 6)

- The updateStartPositions updates startY and startTouchY:

```
public void updateStartPositions( int index, int y ) {
    startY = params[index].topMargin;
    startTouchY = y;
}
```

PuzzleView Class (6 of 6)

- The moveTextViewVertically method moves a TextView by the length of a touch swipe:

```
public void moveTextViewVertically( int index,
                                    int y ) {
   params[index].topMargin = startY + y -
                             startTouchY;
   tvs[index].setLayoutParams( params[index] );
}
```

PuzzleView Version 2

- In Version 2, we place the TextView in the correct slot and we check whether the puzzle is solved.

MainActivity—onTouch (1 of 3)

- When the user releases the TextView (UP action):

```
case MotionEvent.ACTION_UP:
   // move is complete: swap the 2 TextViews
  // if user won, disable listener
```

MainActivity—onTouch (2 of 3)

- The move is complete: swap the two TextViews.

```
int newPosition = puzzleView.tvPosition(
    index );
puzzleView.placeTextViewAtPosition( index,
    newPosition );
```

- We need to add the two methods above to PuzzleView.

MainActivity—onTouch (3 of 3)

- If the user just won, disable the listener to stop the game.

```
if( puzzle.solved(
    puzzleView.currentSolution( ) ) )
      puzzleView.disableListener( );
```

- We need to add the two methods above to the PuzzleView class.

PuzzleView Class (1 of 7)

- We add two instance variables:

```
private int emptyPosition;
private int [ ] positions;
```

PuzzleView Class (2 of 7)

- emptyPosition (an index) stores the y position of the “empty position”, where the TextView the user is currently moving was before the move.
- The positions array stores the y positions of all the elements in tvs.

PuzzleView Class (3 of 7)

- We instantiate positions inside the buildGuiByCode method.
- Inside fillGui, we initialize all the values of the positions array:

```
positions[i] = i
```

PuzzleView Class (4 of 7)

- Inside updateStartPositions, we need to update emptyPosition:

```
emptyPosition = tvPosition( index );
```

PuzzleView Class (5 of 7)

- The disableListener method un-registers all the TextViews on a listener (this MainActivity when the method is called):

```
public void disableListener( View.OnTouchListener
        listener ) {
   for( int i = 0; i < tvs.length; i++ )
      tvs[i].setOnTouchListener( null );
}
```

PuzzleView Class (6 of 7)

- The tvPosition method returns the position index within the screen of the TextView at a given index within the array tvs:

```
// Accuracy is half a TextView's height
public int tvPosition( int tvIndex ) {
    return ( params[tvIndex].topMargin +
        labelHeight/2 ) / labelHeight;
}
```

PuzzleView Class (7 of 7)

- The placeTextViewAtPosition swaps the TextView tvs[tvIndex] and tvs[positions[toPosition]]:

```
public void placeTextViewAtPosition( int tvIndex,
        int toPosition ) {
  // Move current TextView to position position
  // Move TextView just replaced to empty spot
  // Reset positions values
}
```

Moving I LOVE to the MOBILE Position

Index of TextView	Text	Position	toPosition	tvIndex	emptyPosition
1	MOBILE	0	0		
0	PROGRAMMING	1			
2	USING	2			
4	I LOVE	3		4	3
3	JAVA	4			

placeTextViewAtPosition

- Move current TextView at index tvIndex to position toPosition:

```
params[tvIndex].topMargin
    = toPosition * labelHeight;
tvs[tvIndex].setLayoutParams( params[tvIndex] );
```

placeTextViewAtPosition (1 of 3)

- Move TextView just replaced to empty spot:

```
        int index = positions[toPosition];
        params[index].topMargin
            = emptyPosition * labelHeight;
        tvs[index].setLayoutParams( params[index]
);
```

placeTextViewAtPosition (2 of 3)

- Reset positions values:

```
positions[emptyPosition] = index;
positions[toPosition] = tvIndex;
```

placeTextViewAtPosition (3 of 3)

- The currentPosition method returns the current user solution as an array of Strings:

```
public String [ ] currentSolution( ) {
  String [] current = new String[tvs.length];
  for( int i = 0; i < current.length; i++ )
    current[i] = tvs[positions[i]].getText( ).toString( );
   return current;
}
```

Puzzle App, Version 3 (1 of 2)

- In Version 3, after the user has successfully solved the puzzle, if the user double taps on MOBILE, we change it to ANDROID.

Puzzle App, Version 3 (2 of 2)

- First, we explore swipes and taps.
- The GestureDetector class, along with its static inner interfaces, GestureDetector.OnGestureListener and GestureDetector.OnDoubleTapListener, provide the tools and functionality for gestures and taps.

Touches, Version 2

- We first practice with those two interfaces.
- GestureDetector.OnGestureListener contains callback methods for gestures.
- GestureDetector.OnDoubleTapListener contains callback methods for taps.
- We use our Practice Touch app, Version 2.

GestureDetector.OnGestureListener

- Has six callback methods:
  - onDown, onFling, onLongPress, onScroll, onShowPress, onSingleTapUp

GestureDetector.OnDoubleTapListener

- Has three callback methods:
  - onDoubleTap, onDoubleTapEvent, onSingleTapConfirmed

GestureDetector (1 of 4)

Method	Description
GestureDetector(Context context, GestureDetector.OnGestureListener gestureListener)	Creates a GestureDetector object for the context and using gestureListener as the listener called for gesture events. We must use this constructor from a User Interface thread.
void setOnDoubleTapListener(GestureDetector.OnDoubleTapListener doubleTapListener)	Sets doubleTapListener as the listener called for double tap and related gestures.
boolean onTouchEvent(MotionEvent e)	Called when a touch event occurs; triggers a call to the appropriate callback methods of the GestureDetector.OnGestureListener interface.

GestureDetector (2 of 4)

- We define a listener class that implements the GestureDetector.OnGestureListener and GestureDetector.OnDoubleTapListener.
- We create a listener object of that class.
- We instantiate a GestureDetector object, using the listener object.

GestureDetector (3 of 4)

- We set handler as the object listening to tap events (if needed/appropriate).
- Inside the onTouchEvent method of the Activity class, we call the onTouchEvent method of the GestureDetector class with the GestureDetector instance variable.
- This triggers the dispatching of the touch event: depending on the touch event, the appropriate method of the handler class will be automatically called.

GestureDetector (4 of 4)

- See Example.
- All the nine methods are implemented with a Logcat statement inside saying they are called.

```
public class MainActivity extends AppCompatActivity
  implements GestureDetector.OnGestureListener,
  GestureDetector.OnDoubleTapListener {
  public static final String MA = "MainActivity";
  private GestureDetector detector;

  protected void onCreate( Bundle savedInstanceState ) {
    super.onCreate( savedInstanceState );
    detector = new GestureDetector( this, this );
    detector.setOnDoubleTapListener( this );
  }

  public boolean onTouchEvent( MotionEvent event ) {
    Log.w( MA, "Inside onTouchEvent" );
    detector.onTouchEvent( event );
    return true;
  }

  public boolean onFling( MotionEvent e1, MotionEvent e2,
                          final float velocityX, final float velocityY )  {
    Log.w( MA, "Inside onFling" );
    return true;
  }

  public boolean onDown( MotionEvent e ) {
    Log.w( MA, "Inside onDown" );
    return true;
  }

  public void onLongPress( MotionEvent e ) {
    Log.w( MA, "Inside onLongPress" );
  }

  public boolean onScroll( MotionEvent e1, MotionEvent e2,
                           float distanceX, float distanceY ) {
    Log.w( MA, "Inside onScroll" );
    return true;
  }

  public void onShowPress( MotionEvent e ) {
    Log.w( MA, "Inside onShowPress" );
  }

  public boolean onSingleTapUp( MotionEvent e ) {
    Log.w( MA, "Inside onSingleTapUp" );
    return true;
  }

  public boolean onDoubleTap( MotionEvent e ) {
    Log.w( MA, "Inside onDoubleTap" );
    return true;
  }

  public boolean onDoubleTapEvent( MotionEvent e ) {
    Log.w( MA, "Inside onDoubleTapEvent" );
    return true;
  }

  public boolean onSingleTapConfirmed( MotionEvent e ) {
    Log.w( MA, "Inside onSingleTapConfirmed" );
    return true;
  }
}
```

Test: Single Tap

- Output is:

  Inside onTouchEvent

  Inside onDown

  Inside onTouchEvent

  Inside onSingleTapUp

  Inside onSingleTapConfirmed

- If we want to handle single tap processing, we place our code inside onSingleTapConfirmed.

Test: Double Tap

- Output is:

  Inside onTouchEvent
  Inside onDown
  Inside onTouchEvent
  Inside onSingleTapUp
  Inside onTouchEvent
  Inside onDoubleTap
  Inside onDoubleTapEvent
  Inside onDown
  Inside onTouchEvent
  Inside onDoubleTapEvent

Processing Taps (1 of 2)

- We can place code inside onSingleTapUp and onDoubleTapEvent (called twice on DOWN and UP action).
- If we want to process a double tap only, we place our code inside onDoubleTapEvent.

Processing Taps (2 of 2)

- Triple tap = double tap followed by a single tap.
- Quadruple tap = double tap followed by a double tap.

Test: Swipe

Inside onTouchEvent
Inside onDown
Inside onTouchEvent
Inside onScroll
Inside onTouchEvent
Inside onScroll
Inside onTouchEvent
Inside onScroll
Inside onTouchEvent
Inside onFling

onFling Method

```
public boolean onFling( MotionEvent e1, MotionEvent e2,
        final float velocityX, final float velocityY ) {
  // we can access x and y coordinates of e1 and e2
  // we can also get a time reference for both
  // (call getEventTime method)
  // velocity parameters are in pixels per second
}
```

Puzzle Version 3 (1 of 8)

After the user solves the puzzle, if the user double taps on MOBILE, it changes to ANDROID.

Puzzle Version 3 (2 of 8)

- We edit the Model (we still keep it very simple).

```
public String wordToChange( ) {
     return "MOBILE";
}

public String replacementWord( ) {
    return "ANDROID";
}
```

Puzzle Version 3 (3 of 8)

- GestureDetector.SimpleOnGestureListener is a convenient class that implements the GestureDetector.OnGestureListener and the GestureDetector.OnDoubleTapListener interfaces with do nothing methods returning false.
- It is convenient if we are only interested in one or two methods (we do not have to implement all nine).

Puzzle Version 3 (4 of 8)

- MainActivity already extends AppCompatActivity, so it cannot extend another class.
- We code a private class inside MainActivity that extends GestureDetector.SimpleOnGestureListener and implement its onDoubleTapEvent method.

Puzzle Version 3 (5 of 8)

```
private class DoubleTapHandler
        extends GestureDetector.SimpleOnGestureListener {
    public boolean onDoubleTapEvent( MotionEvent event ) {
      // code to handle double tap here
   }
}
```

Puzzle Version 3 (6 of 8)

```
private class DoubleTapHandler
        extends GestureDetector.SimpleOnGestureListener {
    public boolean onDoubleTapEvent( MotionEvent event ) {
        // retrieve y coordinate of double tap
        // retrieve index of TextView for the y coordinate
        // retrieve the text of that Text view
        // if the text is the one to change, change it
    }
}
```

Puzzle Version 3 (7 of 8)

- We retrieve y coordinate of double tap:

  int touchY = (int) event.getRawY( );

- Note that the y coordinate is within the activity's view that is the whole screen.
- Thus, getY and getRawY return the same value
- The y coordinate of the touch within puzzleView is touchY - actionBarHeight – statusBarHeight.

Puzzle Version 3 (8 of 8)

- We retrieve the index of the TextView for the y coordinate.

```
int index = puzzleView.indexOfTextView( touchY
            - actionBarHeight - statusBarHeight );
```

- We change the text if it is the correct TextView:

```
if( puzzleView.getTextViewText( index )
      .equals( puzzle.wordToChange( ) ) )
          puzzleView.setTextViewText( index,
              puzzle.replacementWord( ) );
return  true;
```

PuzzleView Class (1 of 3)

- Now we need to add the indexOfTextView, getTextViewText, and setTextViewText methods to the PuzzleView class.

```
public int indexOfTextView( int y ) {
    int position = y / labelHeight;
    return positions[position];
}
```

PuzzleView Class (2 of 3)

```
public String getTextViewText( int tvIndex ) {
  return tvs[tvIndex].getText( ).toString( );
}
```

PuzzleView Class (3 of 3)

```
public void setTextViewText( int tvIndex,
        String s ) {
    tvs[tvIndex].setText( s );
}
```

Puzzle App, Version 4

- In Version 4, we optimize (i.e., maximize) the size of the font in the TextViews.
- Currently, it is 32 (in styles.xml).
- We build a utility class with one static method that sets the font size of a TextView to a maximal value so that its fits on one line inside the TextView.

DynamicSizing Class

- We name that utility class DynamicSizing.
- We name the static method setFontSizeToFitInView.

Puzzle App, Version 4 (1 of 4)

```
public class DynamicSizing {
  public static int setFontSizeToFitInView( TextView tv ) {
    …
    // returns the font size
```

Puzzle App, Version 4 (2 of 4)

- Inside the fillGui method of PuzzleView, we size the font in the TextViews so that it is optimal (i.e., maximal) amd still fits on one line.
- We use the same font size for all the TextViews.

Puzzle App, Version 4 (3 of 4)

```
  // inside for loop
  tvs[i].setWidth( params[i].width );
  tvs[i].setPadding( 20, 5, 20, 5 );
  // find font size dynamically
  int fontSize =
     DynamicSizing.setFontSizeToFitInView( tvs[i] );
  if( minFontSize > fontSize )
     minFontSize = fontSize;
}
```

Puzzle App, Version 4 (4 of 4)

```
// after the for loop
// set font size for all the TextViews
for( int i = 0; i < tvs.length; i++ )
      tvs[i].setTextSize(
TypedValue.COMPLEX_UNIT_SP, minFontSize );
```

Touches and Gestures

- Touches, gestures, taps
- Capturing and handling these events
- Single vs double vs triple tap ...