CMPT 381 Assignment 3
Android Basics, Touch Interaction, Multiple Views

Due: Friday, March 13, 11:59pm (handin closes Sunday, March 15, 11:59pm)

Overview

In this assignment you will build an interactive Android application in stages. The application is a Chart Extractor, used to reverse-engineer the data in an XY chart from a picture of the chart. Part 1 covers Android versions of basic GUI components that have already been introduced in class, such as menus, widgets, layout, and custom drawing; it also introduces a few new Android concepts such as Intents. Part 2 covers the design of interaction techniques for the main screen of the data-extraction application (there are two main interactions: adjusting the chart's bounds, and placing and adjusting the chart points). Part 3 involves a second view that is a scaled version of the main view, to assist the user in making accurate adjustments of the chart points (i.e., overcoming the "fat finger" problem).

Part 1: Android basics for the ChartExtract app

After reviewing the "getting started" Android resources linked from the moodle, use the Android Studio IDE to develop an app named ChartExtract with the following properties:

- Minimum API level: 26 (Android 8.0)
- Create the app as an "Empty Activity" in the setup wizard
- Project name: ChartExtract
- Language: Java (not Kotlin)

Your app will have the following main components:

- A main view that has a large panel for showing the chart and working with chart points, a detail panel at top left, and a chart-axis panel to let the user specify the chart scale (and see the current location in that scale).
- A menu that has three menu items: "Choose Chart", "Screenshot", and "Export Data"
- An MVC architecture including an interaction model and a state machine for your controller

Notes:

- The app will be run on the Android emulator, using a Pixel 3 profile running API level 26 or later.
- No physical device is needed
- The app will only be run in portrait mode; you do not have to handle orientation changes.

Part 1A: Menu

- Create a file main_menu.xml in the project folder app/res/menu/, and open the file in the editor (by double clicking) to add items to the menu.
    - See pages 135-137 of "Learn Android Studio 3 : Efficient Android App Development" (moodle)
    - See developer.android.com/guide/topics/ui/menus
    - For a general introduction to accessing resources in Android (what does "R.menu.main_menu" mean?), see developer.android.com/guide/topics/resources/providing-resources
- Add the method onCreateOptionsMenu(Menu menu) to your main activity
    - See example code at developer.android.com/guide/topics/ui/menus
- Add the method onOptionsItemSelected(MenuItem item) to your main activity
    - See example code at developer.android.com/guide/topics/ui/menus
- Add code to the switch statement in onOptionsItemSelected to handle the different menu items
- Note that the handlers for menu events should be in your main activity; they do not have to go through a Controller class

Part 1B: Chart Chooser

- Drag the example chart (sample-chart.png from the Assignment folder on the moodle) onto your emulator (it should be saved to the device's Downloads folder; you can inspect the file system with the Files app that is installed on the emulator)
- Add a method in your main activity that responds to selection of the "Choose Chart" menu item (see above)
- Create and start an Intent to use Android's built-in content chooser activity.
    - Example code for starting the chooser activity:

```
Intent intent = new Intent();
intent.setType("image/*");
intent.setAction(Intent.ACTION_GET_CONTENT);
startActivityForResult(Intent.createChooser(intent, "Select a chart"), 123);
```

- In your main activity, create method protected void onActivityResult(int requestCode, int resultCode, Intent data) to respond to the chooser activity
    - Example code for handling the result from the chooser activity:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if(requestCode == 123 && resultCode == RESULT_OK) {
        Uri selectedImage = data.getData(); //The uri with the location of the image
        // do something with the selected image
    }
}
```

- Your app should then pass the Uri object (corresponding to the chart image) to the chart view for display
- Note that you can create a Bitmap from a Uri using:

```
MediaStore.Images.Media.getBitmap(parentContext.getContentResolver(), imageUri);
```

Part 1C: Main Screen

- In your main activity, build the main screen of the app using three panels:
- A custom main panel at the bottom (class ChartView), where the chart will be displayed and user interactions will occur
- A custom detail panel at top left (class DetailView) that shows details when the user positions a chart point
- A custom chart-axis panel at top right (class AxisView) that lets the user set up the chart's scale and see the current point's location within that scale
- When the user chooses a chart (see above), that chart should be displayed in the ChartView, scaled to fit the view (the chart may look squashed; this is fine).

Part 1D: ChartView

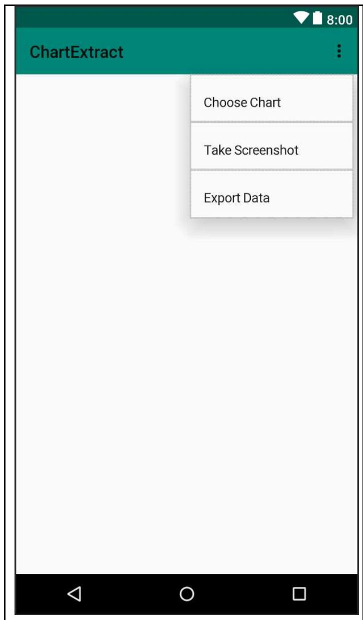
- Develop a custom view (ChartView extends View) to show the chart
- The chart view should display the chosen chart using Android's 2D graphics capabilities
    - See developer.android.com/training/custom-views/custom-drawing
    - See the "drawBitmap" methods of class Canvas: developer.android.com/reference/android/graphics/Canvas.html
- The chart view will later be used to show the chart points and the bounds rectangle (see Part 2)

Part 1E: DetailView

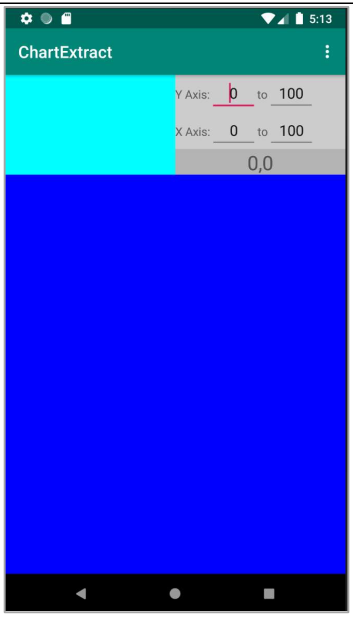
- Develop a custom view (DetailView extends View) to show a portion of the chart
- In Part 3, you will use this view to show a magnified view around the user's current manipulation point
- For now, the detail view should just display the chosen chart using Android's 2D graphics capabilities, scaled to fit the size of the detail view (see pictures below)

Part 1F: AxisView

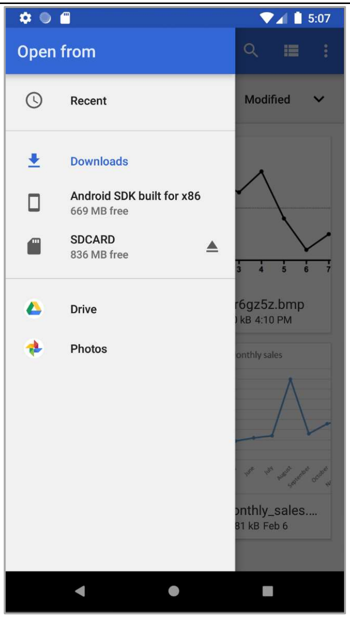
- Develop a custom view (AxisView extends LinearLayout) to allow the user to set the scales of the X and Y axes of the chart (i.e., the start and end numbers that are in the chart's axes).
- Use classes TextView and EditText for the labels and editable text fields
- At the bottom of this view, add an additional label that will display the user's current location
- Introduction to LinearLayout in Android: developer.android.com/guide/topics/ui/layout/linear



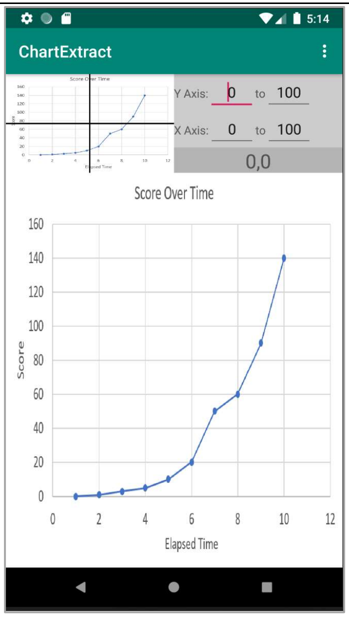
Options menu as shown in the IDE's menu editor



Main screen, with chart view (blue), detail view (cyan), and axis view (grey)



Chooser activity



Main screen after choosing chart (Part 1 only)

Part 2: Setting the chart's bounds, and adding chart points

The second part of the assignment involves adding points to the view that can be used to extract data from the chart picture. This involves two stages: first, the bounds of the chart area must be specified (so that the system can interpolate chart points); and second, points must be added to the view and placed over the data points in the chart.

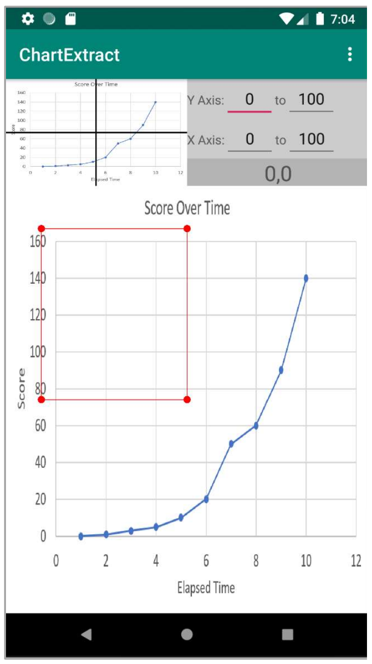
Part 2A: The Bounds Rectangle

In addition to displaying the chart itself, draw a red rectangle on the screen that allows the user to graphically indicate the bounds of the chart area. The corners of the rectangle should show selection handles that the user can grab and drag to change the rectangle's location and extents.

The size and location of the bounds rectangle will be used (along with the axis limits from the axis view) to calculate the true locations of the chart points.

Requirements for the bounds rectangle:

- The rectangle's data should be stored in the InteractionModel
- Grabbing one of the rectangle's handles should allow for a fat-finger tolerance (i.e., the user's touch point only needs to be close enough to a handle rather than directly on the handle). How close is "close enough" is up to you.



Bounds rectangle in red with handles for resizing

- Once the user has grabbed a handle, the rectangle resizes dynamically as they drag the handle.

Part 2B: Chart Points

When the user touches and releases on the main view, the system creates a "chart point" – a graphical object that represents one data point in the chart, and that can be dragged by the user to adjust the point. The purpose of the chart points is to extract specific data points that appear in the chart picture: for each data point that the user wants to extract, they will create a chart point and place it on the picture.

If the user touches near to an existing chart point, then that point becomes the selected point, and is dragged as the user's finger moves on the screen. When the user lifts their finger, the point is no longer selected (i.e., there is no persistent selection). Use the same idea of "close enough" as described above to allow people to select points without needing to be exactly accurate.

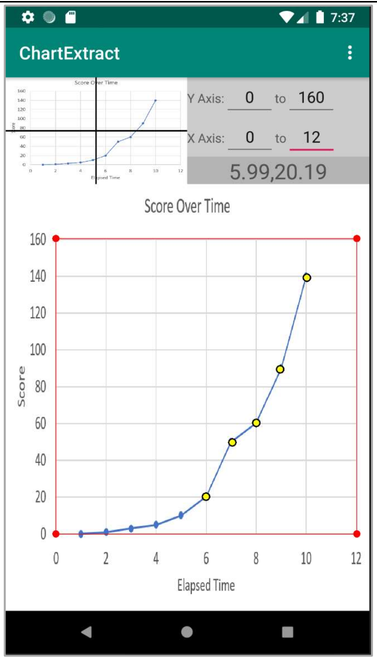
Whenever a chart point is selected, the axis view shows the location of the chart point in the scale of the chart (as defined by the bounds rectangle and the axis endpoints set by the user). In the picture at right, the selected chart point is at 5.99 on the chart's X axis, and 20.19 on the chart's Y axis.

Part 3: The DetailView

The third part of the assignment uses the detail view to provide a magnified version of the area around the current selection, to help the user position a chart point accurately.

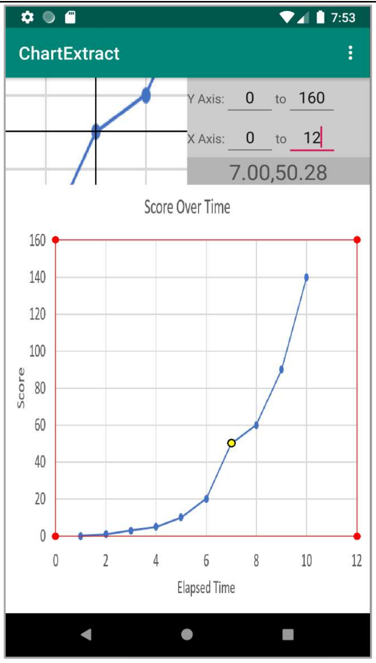
Whenever a chart point is selected, the detail view will show a 2X magnification of the original chart image, centred at the location of the chart point. This means that as the user drags a chart point, the detail view will pan to keep the chart point at the centre of the view.

You will also draw cross-hair markers to show the user the exact center of the view (this corresponds exactly to the chart point's location). The cross hairs allow precise positioning of a chart point over the centre of a data marker in the chart.



Y Axis: 0 to 160
X Axis: 0 to 12
5.99,20.19

Bounds rectangle has been set to the chart bounds, and five chart points (yellow circles) have been added. The user's finger is touching the leftmost chart point. Axis view shows the selected chart point's location (in chart scale)



Y Axis: 0 to 160
X Axis: 0 to 12
7.00,50.28

Detail view showing magnified version of the chart directly under the selected chart point.

Part 4: Exporting Data and Saving Screenshots

Part 4A: Screenshot

When the user selects the "Take Screenshot" menu item, the system will write a bitmap of the chart view to the file system. Use the following example code for your menu item handler:

```
Intent intent2 = new Intent(Intent.ACTION_CREATE_DOCUMENT)
        .addCategory(Intent.CATEGORY_OPENABLE)
        .setType("image/png")
        .putExtra(Intent.EXTRA_TITLE, "screenshot.png");
startActivityForResult(intent2, 234);
```

- In your main activity's onActivityResult method, use the following example code to write the file:

```
        if (requestCode == 234 && resultCode == RESULT_OK) {
            Uri newFile = data.getData();
            try {
                ParcelFileDescriptor pfd =  getContentResolver().openFileDescriptor(newFile, "w");
                FileOutputStream fos = new FileOutputStream(pfd.getFileDescriptor());
                Bitmap screen = Bitmap.createBitmap(chart.getWidth(), chart.getHeight(),
                                 Bitmap.Config.ARGB_8888);
                Canvas canvas = new Canvas(screen);
                myChartView.draw(canvas);
                screen.compress(Bitmap.CompressFormat.PNG, 100, fos);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
```

Part 4B: Export Chart Data

When the user selects the "Export Data" menu item, the system will write a comma-separated values file (CSV) containing each chart point's X and Y coordinates (in chart scale). Use the same basic process as described above for creating and writing the file, but write a simple text file instead of an image file.

- For an example of writing a text file, see p. 199 of "Learn Android Studio 3 : Efficient Android App Development" (moodle)

What to hand in

This assignment is to be done individually; each student will hand in an assignment.

- Hand in a zip file of your Android Studio project folder and a readme.txt file that indicates exactly what the marker needs to do to run your code.
- Note that some specifications of Parts 2 and 3 supersede those of Part 1 (e.g., what appears in the detail view)

Where to hand in

Hand in your zip file and readme.txt to the link on the course moodle.

Evaluation

Marks will be given for producing a system that meets the interaction requirements above, that uses an appropriate architecture, that is well organized at the code level, and that compiles and runs without errors. Note that no late assignments will be allowed, and no extensions will be given, without medical reasons. Weighting of the parts in the overall grade: Part 1=40%, Part 2=30%, Part 3=20%, Part 4=10%