Mobile Application Programming: Android

CS4962 Fall 2016
Project 2
Due: 11:59PM Monday, October 3rd

Abstract

Using Project 1 as a base, create a painting application that allows the user to paint a pictures and save them in a collection. The paintings can be further updated by navigating to the desired painting and touching it to add paint, just as creating the painting from scratch. The application should save the paintings across runs of the application, establishing a persistent store of paintings. The UI of the application will be modified to allow it to be used naturally in landscape orientation and on tablet devices. The paintings will also have a “How it was Painted” feature. This feature will take the form of a play / stop button which, when pressed, will toggle from play to stop and make all UI elements except the button inactive. For the next 5 seconds, the currently selected painting will be erased, then drawn one stroke at a time until it takes its fully painted form. Your application should also support rotation of the device.

Components

- **Side Menu:** A strip along any edge of the interface containing buttons to activate actions. The menu will contain 4 buttons. The first two will depict back and forward arrows. Pressing the back arrow will allow the user to access paintings created previously. Pressing the forward arrow will allow the user to move the opposite direction, until they have reached the current drawing. These two buttons will take on different states, depending on where in the ordered collection of paintings they currently find themselves, described below. The menu will also have the play / stop button, described below, and a control for selecting the current color. The current color “button” does not necessarily have to be an instance of the Button class. It should depict a paint splotch that indicates the currently selected color, in the same style as the color selector from Project 1. This color selector will be found in another Activity, described below.
- **Back / Forward Buttons:** These buttons allow the user to access the drawings in the Gallery. This is an easier alternative to using a ListView to display paintings in a list and selecting them (see Extra Credit). These buttons control an index that indicates the current painting. The index should be able to vary from 0 to the number of paintings in the Gallery (NOT minus 1). If the index is equal to the number of drawings in the Gallery, this indicates that the user will create a new drawing by adding strokes to the paint area view. This will be the case when the app opens for the first time, as the number of paintings in the Gallery will be 0 and the index is initialized to 0. Once the user adds a stroke to the paint area, a painting will be created, the Gallery painting count will now be 1, and the index will continue to be 0, indicating the user is editing the 0-index painting. The Back button should depict a left arrow, and should be disabled when the index is at position 0 and enabled otherwise. The Forward button should depict a right arrow and should be disabled when the index is equal to the number of drawings in the Gallery and enabled otherwise. Both buttons should be disabled while the play function is active.
- **Paint Area View and Play / Pause Button:** Allows painting as described in Project 1. The user can add strokes to the currently selected painting using their finger, which use the current paint color. When the play / stop button is pressed, the button will toggle from a “play” triangle

to a “stop” square, and the paint area view and other UI elements will not respond to touches for 5 seconds. During this time, any strokes that had been added to the painting will be erased, then added one at a time to the painting. If the painting had 10 strokes, a stroke would be added to the painting every 1/2 of a second. If it had 20 strokes, a stroke would be added every 1/4 of a second. When the time period has passed, the painting should have had all strokes restored, and the button should automatically toggle from the “stop” square to the “play” triangle again. If the user taps the play / stop button while the animation is proceeding, make the animation proceed to the end immediately, adding the strokes, restoring the button state to “play”, and re-enabling the other UI elements. This animation is most easily implemented using a ValueAnimator that animates an integer index that ranges from 0 to the number of strokes in the current drawing, and provides the appropriate number strokes to the paint area view each time its animation method is called in the same way as when the user selects another painting in the collection.

- **Current Color Control:** Depicts a paint splotch that indicates the color that will be used to paint when the user touches the paint area. The control should be an instance of the same class as the one used in the color selection palette. The OnClickListener property should be attached to the Activity that is showing the side menu and paint area. When clicked, the Activity should call Activity.startActivity() or Activity.startActivityForResult() with an Intent that indicates that the system open a color selection activity.
- **Color Selection:** The palette view from Project 1 should be moved to its own Activity that is activated by the Current Color Control. This activity should have as its content view a view group containing the palette view and color add / remove knobs and buttons. These should function as they do in Project 1. The currently selected color needs to be transported back to the Current Color Control, which can be challenging as this control is in the view tree managed by another activity now. Use whatever mechanism you’d like to do this. A few ideas: use startActivityForResult and onActivityResult; share a data model between the two activities; use SharedPreferences.

Considerations

- **Architecture:** Using the Model-View-Controller application architecture scheme will be one of the best ways to organize this application.
- **Data Model**(**s**): The application will manage a collection of paintings that need to persist across runs of the application. Use files to store this data to non-volatile storage. You can use any file format you’d like to store the data (e.g. flat binary, text, Java object serialization, XML, JSON) but cannot use 3rd party libraries. Only libraries provided by the Android system and Java runtime should be used to compose the files, with one exception. Those who would like to use GSON to compose a JSON file format are welcome to use it. This is not the only way to make JSON files on Android, but it is a popular one. Note that JSON is probably not the easiest way to do this, but those who already are familiar with it may choose to use it. The paintings the user creates need to be saved, as well as the colors in the palette color selector. You may find it easier to have two data models in your application, as the data managed by these are completely independent of one another.
- **Views**: The data used by the Paint Area View to draw its content should use view-type classes (PointF, the Color class, Path, etc). You are encouraged to not directly use data model objects in your views. Since this application is very view-centric, you will find that doing this will involve significant data duplication. However, the application requirements, particularly the “play” feature, are much more easily implemented if you do so. Adding properties for

currentColor and completedStrokes as well as an OnStrokeCompletedListener to your Paint Area View will make the data conversion more

- **Controllers**: Activity sub-classes will be the primary control mechanisms in the application. These should direct the construction of the user interface and respond to user interaction. The painting activity should contain methods for converting data from PaintAreaView strokes to Drawing strokes and vice-versa. Typically these conversion methods will also perform coordinate system conversions. The strokes in the PaintAreaView will use pixel coordinates, while the data model strokes should use some other coordinate system, like millimeters, device-independent-pixels (1/160 of an inch), or some other display-device-independent coordinate system. Doing this will allow you to show data on a display that is not the same as the display in which the data was created, including a rotated version of the current one! This conversion can maintain aspect ratio, stretch to fill, or stretch to fit, or center the data in order to make it fit on the destination display. The coordinate system conversion defines where and how this happens. The palette activity will be significantly more simple than the painting activity, and may even act more as a dialog if you choose to use two data models in your application.

Extra Credit

- **Drawing List (10%):** Those interested in looking ahead in the class and defining a 3rd Activity that manages an drawing ListView will receive extra credit. This activity will be the entry point to the app and display a list of paintings that have been created so far. The list elements should have a summary of the painting of some kind and, when an element is tapped, will open the painting activity for the selected painting. The summary could be an image of the painting or a text description with the stroke count and creation date of the painting.

Handin

You should hand in your zipped project, including any supporting files, using the CADE Lab handin system on the command line:

handin cs4530 project2 your_project_zip_file.zip