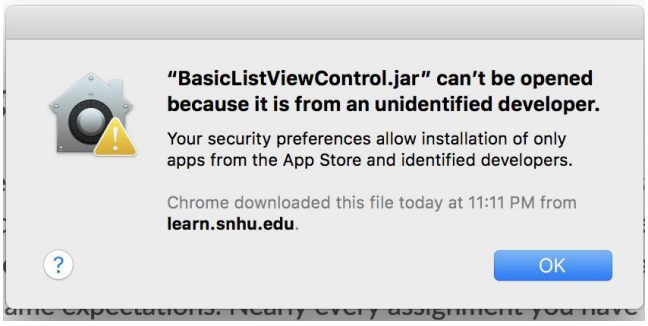
CS 250 Developing Basic ListView Control in Java Tutorial

Previewing the Basic ListView Control

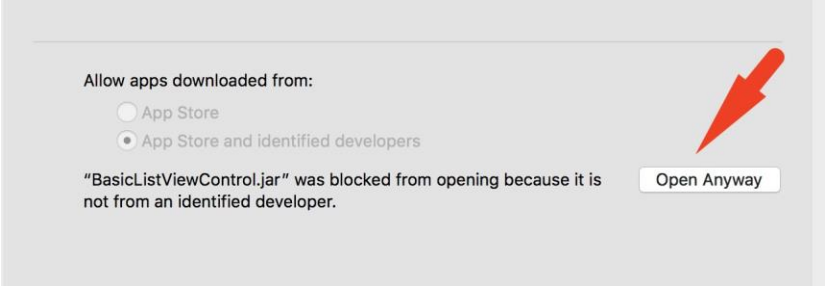
Before developing your code, run the JAR file that you have been given to understand how the Basic ListView Control works. Use the JAR file as a reference to understand the goal, then import the code into Eclipse and make the appropriate modifications.

1. First, download and unzip the SampleListView.zip file, which is linked in Step 1-B of the Module Three Basic ListView Control Assignment Guidelines and Rubric. In the unzipped folder, find and run the BasicListViewControl.jar file.

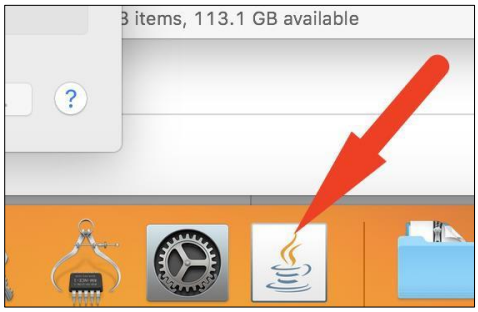
2. If the file runs when you open it, skip ahead to Step 5. If you see an error message, you will need to give the JAR file explicit permissions to run. Follow Steps 3–4 to give the file the necessary permissions.



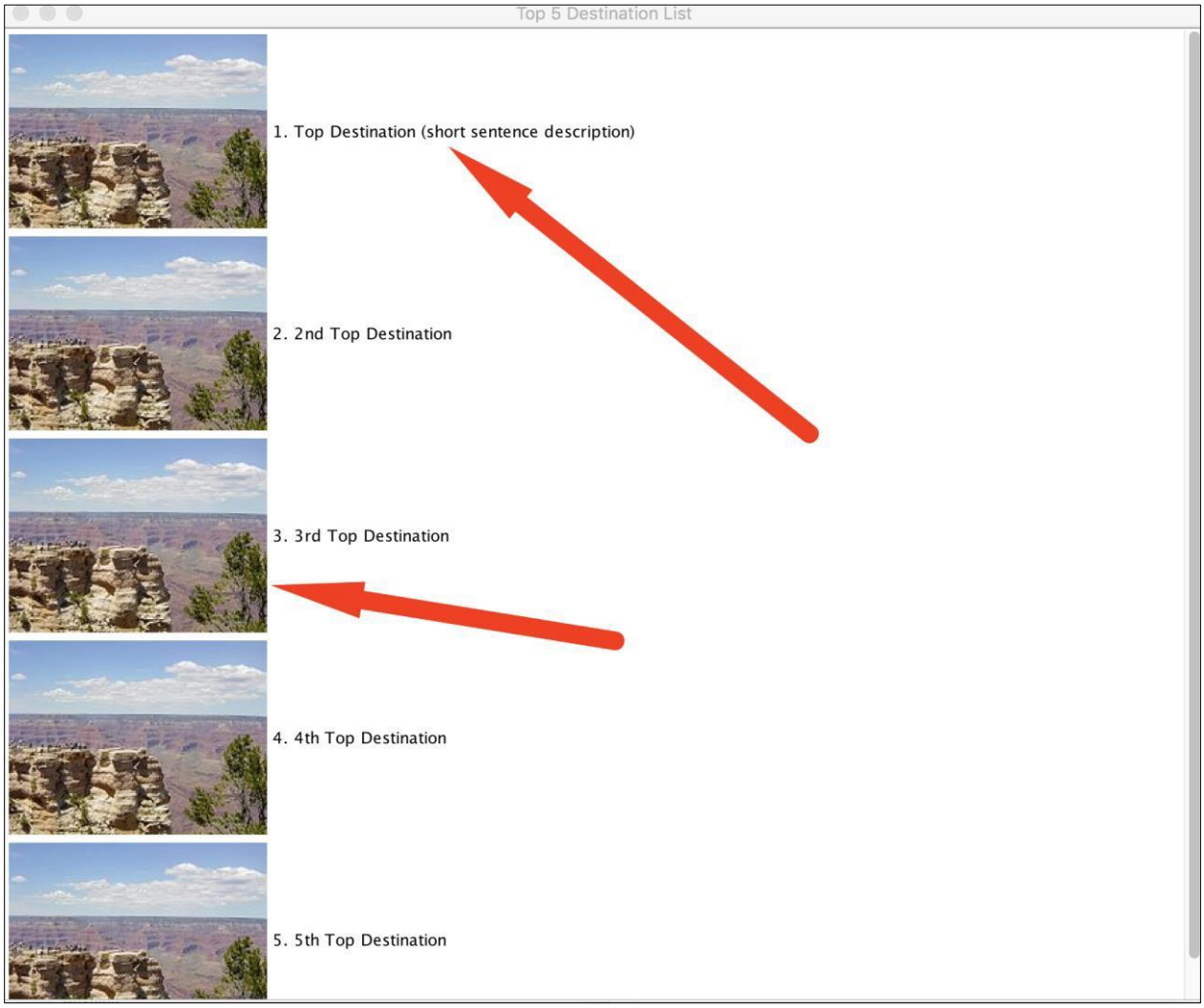
3. On a Mac, go to **System Preferences**, then **Security and Privacy.** Inside that utility, you can give the JAR explicit permissions to run by clicking **Open Anyway**.



4. This will allow the file to launch. You will see a basic Java icon in the tray, and the file should run. Both Windows and Mac will have a similar generic Java icon.



5. You should see a screen with placeholders for the images and labels. Explore the file so that you understand how it works. In the assignment, you will be modifying the code by finding five destination images to replace the placeholder images, updating the labels, and making one other style change to the ListView. You should be sure to comment your code.



6. To modify the code, import the zipped folder into the Codio virtual lab. Open Eclipse and import the source code. Inside Eclipse, open the **TopFiveDestinationList.java** file and find the

TopDestinationListFrame. Code examples will not be provided, but you are encouraged to look through this section of code for possible modifications.

```
class TopDestinationListFrame extends JFrame {
    private DefaultListModel listModel;

    public TopDestinationListFrame() {
        super("Top Five Destination List");

        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setSize(900, 750);

        listModel = new DefaultListModel();


        //Make updates to your top 5 list below. Import the new image files to resources directory.
        addDestinationNameAndPicture("1. Top Destination (short sentence description)", new ImageIcon(getClass().getResource("/resources/TestImage.jpg")));
        addDestinationNameAndPicture("2. 2nd Top Destination", new ImageIcon(getClass().getResource("/resources/TestImage.jpg")));
        addDestinationNameAndPicture("3. 3rd Top Destination", new ImageIcon(getClass().getResource("/resources/TestImage.jpg")));
        addDestinationNameAndPicture("4. 4th Top Destination", new ImageIcon(getClass().getResource("/resources/TestImage.jpg")));
        addDestinationNameAndPicture("5. 5th Top Destination", new ImageIcon(getClass().getResource("/resources/TestImage.jpg")));

        JList list = new JList(listModel);
        JScrollPane scrollPane = new JScrollPane(list);

        TextAndIconListCellRenderer renderer = new TextAndIconListCellRenderer(2);

        list.setCellRenderer(renderer);

        getContentPane().add(scrollPane, BorderLayout.CENTER);
    }

    private void addDestinationNameAndPicture(String text, Icon icon) {
        TextAndIcon tai = new TextAndIcon(text, icon);
        listModel.addElement(tai);
    }
```

Customizing the Basic ListView Control

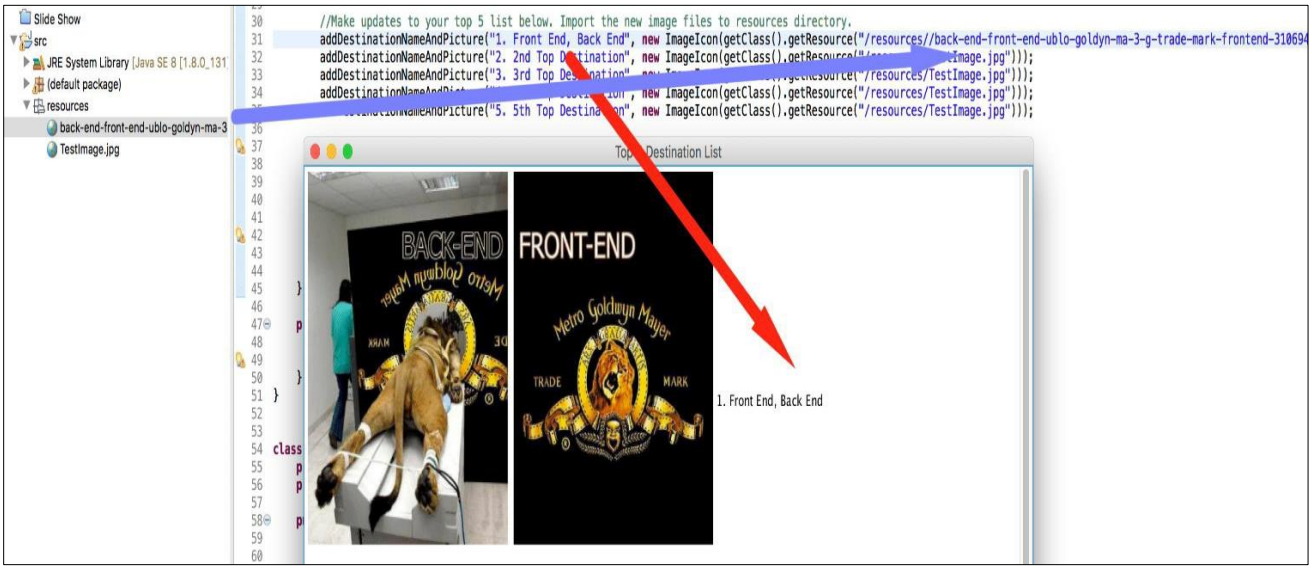
For the programming task, use the source code given to update the simple list to your top five chosen destinations. Include the destination title, a short description, and a small image.

1. A Java UI is a collection of objects that are composed in a hierarchy. What does that mean? There is a main container, the JFrame that holds UI objects. A UI object is a JLabel, text field, radio button, and so on. A UI object is one of the visual elements you see in the frame.

   The hierarchy part means that containers like the JFrame can have sub-containers like panels. The JFrame is like a painter's canvas, and sub-containers would be like boxes that you paint on the canvas to sub-divide the picture.

   The main canvas is the **TopDestinationListFrame,** which is a JFrame. Any objects you want to add to the UI will be on the **TopDestinationListFrame**. There are a handful of objects already on the JFrame. There is a scroll pane, which contains a list, which is comprised of a collection of **TextAndIcon** objects.

   To update the simple list, update the individual **TextAndIcon** objects before they get added to the list, or **listModel**.



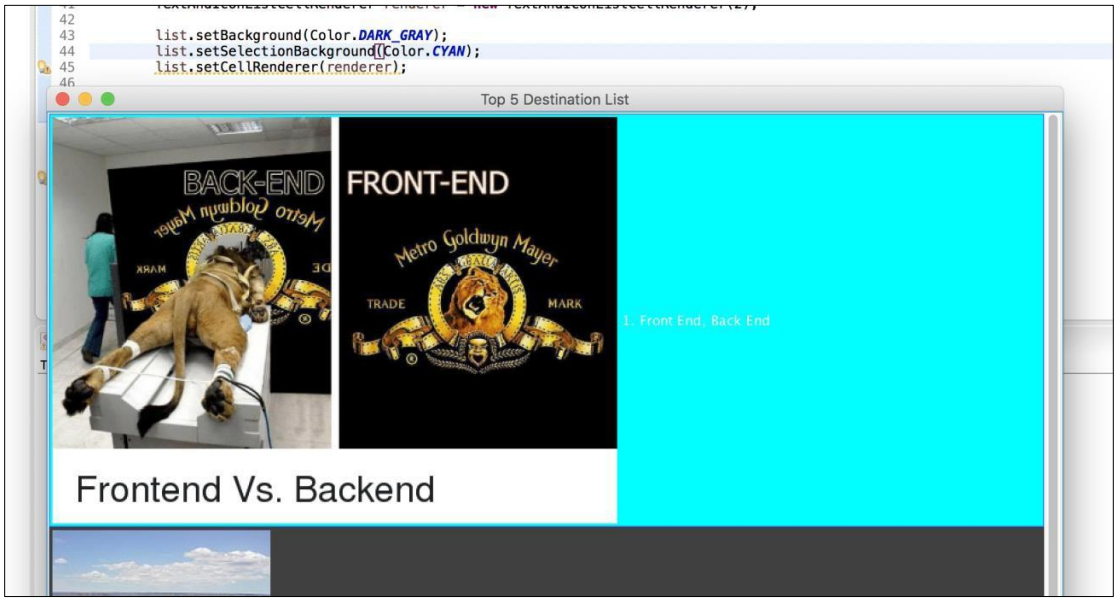
Using this pattern, you can add an updated image and text with a **<Title> - <Description>** pattern. That is all well and good, but there are additional fuzzy requirements. They are fuzzy because of overloaded language.

2. The UI is a collection of objects. Objects can contain objects in a nested pattern. Objects also have attributes that define how to draw them on the screen. For JLabels and text objects, this may be a font or text color attribute. One of your requirements for the assignment is to make a customization. The customization will come in the form of modifying an object's attributes.

   A scroll pane is covering the entire JFrame, so if you try to make modifications to the JFrame or add elements, they will not be visible. Why? Because the scroll pane is nested in the JFrame and covers the entire frame, which is filled with a JList of JLabels. What you have is a screen that is a list of cells. If you want to see changes, modify the attributes of the objects which are visible:

   JFrame -> ScrollPane -> JList -> listModel

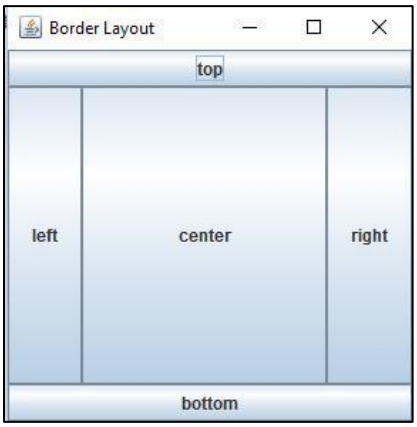
   The listModel is just a collection of text and images, so the object that has visible attributes that you will see is the JList. You will need to modify the JList.



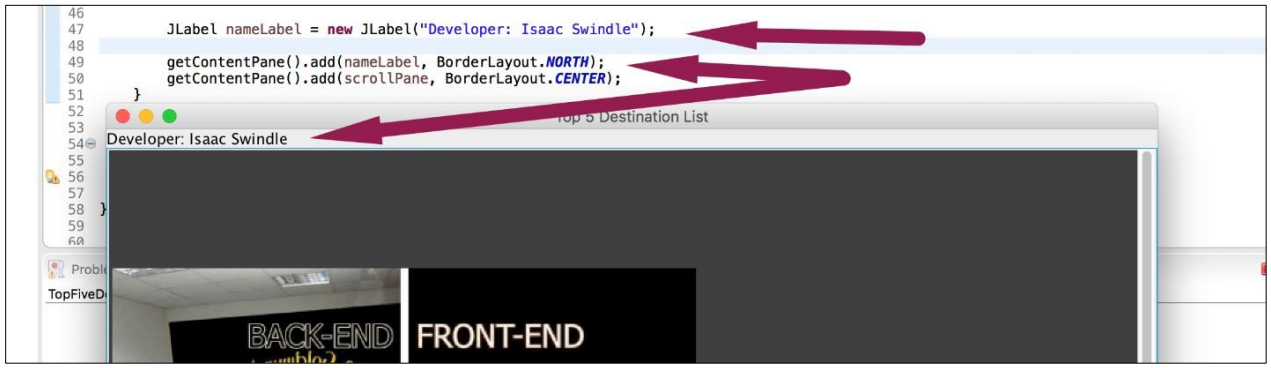
3. You will notice that something must be happening that is preventing a label from being seen. The issue is a simple line of code that is causing the content to take up the entire screen:

   getContentPane().add(scrollPane, BorderLayout.*CENTER*);

   The **BorderLayout.CENTER** constraint is being displayed like this:



4. To add a JLabel above the list, you need to make sure that it will be laid out above the scroll pane. To do this, add a JLabel to the JFrame and set it to **BorderLayout.NORTH** (top).



What you have done here is instantiated a new JLabel object and added it to the JFrame by calling **getContentPane (component, constraint)**. Feel free to play around with these patterns and customize and update as you like.