I'm not robot
reCAPTCHA
Continue

Android graph plotting library

View on GitHub Installation Chart Types Running Demos Contacts Links License Graphs and data plots are wonderful tools for illustrating relationships, representing data trends, and tracking goals in your Android applications. I saw it in person several years ago, when a former student of mine won first place in a student mobile app competition sponsored by the Charleston Defense Contractors Association. A key feature of the winning app, Diabetes and Me, was the ability to chart daily sugar levels. As another example, consider a weight monitoring application that tracks progress against an objective weight. Figure 1 illustrates what such an application might look like on an Android phone. The data uses a red line chart to show monthly average weights for the year 2017. Shows the weight of the lens as a green straight line near the bottom. Although the data values shown in the line chart are hypothetical, they are realistic as far as the author of this article is concerned. John I. Moore Figure 1. Weight monitoring for the year In this article I will use my open source library, GraphLib, to demonstrate the basics of math graphics in Android. It's not the same chart library that my student used for his application. In fact, it is much simpler and easier to use. download Get the source code for the open source Android graphics library introduced in this article. Created by John I. Moore. GraphLibGraphLib overview consists of an interface and eight classes. Three of these classes are internal to the library and only have access to packages, so you won't need to understand them to use GraphLib. Two of the remaining classes have very simple features and the rest is not difficult to collect. Below I will describe the GraphLib interface and each of its eight classes. Note that I used Java 8 features such as functional interfaces and lambda expressions to develop and test the library, but it is relatively simple to modify these features for previous versions of the Java functional interface.GraphLibAs shown in Listing 1, interface Function has only one abstract method and is, therefore, a functional interface. Note that this interface is approximately equivalent to The DoubleUnaryOperator of Java 8, which is present in the java.util.function package. The difference is that Function does not use any Java 8 functionality other than @FunctionalInterface. Removing this annotation is the only change needed to make the Function interface compatible with earlier versions of Java. List 1. interface Function package com.softmoore.android.graphlib; @FunctionalInterface public interface function { public double apply(double x); } The GraphLibClasses Point and Label classes are simple: Point encapsulates a pair of double values representing a point in the x,y plane, and Label encapsulates a double value and a string, where the double value represents a point on an axis and the string is used to label that point. The example in Figure 1 uses points for the line chart and labels for the bottom axis, showing abbreviations of a letter for months. I will give other examples that illustrate the use of these classes later in the article. The GraphFunction, GraphPoints, and ScreenPoint classes are not only very simple, but are also internal to the library and only have access to packages. You don't really need to understand these classes to use the library, so I'll briefly describe them here: GraphFunction encapsulates a function (i.e. a class that implements the interface function) and a color used to draw that function. GraphPoints encapsulates a list of points along with a color used to track them. This class is used internally for both plot points and drawing line charts. ScreenPoint encapsulates a pair of integer values that represent pixel coordinates on an Android device screen. This class is similar but simpler than the Android Point class in the android.graphics package. I provided the source code for these classes in case you are interested in the details. The three remaining classes in the GraphLib library are Graph, Graph.Builder, and GraphView. It is important to understand the role that each of them plays in an Android application. Class Graph contains information about the colors, points, labels, charts, etc., to draw, but is essentially independent of Android graphic details. Although Graph contains many fields, all of them have default values, so you should use the Builder template to instantiate this class. Class Graph contains a nested static subclass named Builder, which is used to create Graph objects. The two graph and graph.builder classes go together, from a developer's point of view, and should be understood, essentially, as one. In fact, you just need to figure out how to use the nested class builder to create a Graph object. Developers really don't do anything directly with a Graph object after it's created but pass it to a GraphView object, which does the job of displaying everything on an Android device. List 2 summarizes the methods available in the Graph.Builder class. Later examples will show you how to use the Generator template to create Graph objects. For now, just note that in addition to the default constructor (first line in list 2) and the build() method (last row in List 2), all other methods return the Builder object. This allows you to concatenate calls to builder methods. List 2. Summary of methods in the Graph.Builder public Builder() public Builder addFunction(Function), int graphColor) public Builder addFunction(Function function) public Builder addPoints(Point[] points, int pointColor) public Builder addPoints(List<Point> points, int pointColor) public Builder addPoints(Point[] points) public Builder addPoints(List<Point> points, int pointColor) public Builder addPoints(Point[] points) public Builder addPoints(List<Point> points) public Builder addLineGraph(Point[] points, int lineGraphColor) public Builder addLineGraph(List<Point> points, int lineGraphColor) public Builder</Point> </Point> </Point> </Point> points) public Builder addLineGraph(List<Point> points) public Builder setBackgroundColor(int bgColor) public Builder setAxesColor(int axesColor) public Builder setFunctionColor(int functColor) public Builder setPointColor(int pointColor) public Builder setWorldCoordinates(double xMin, double xMax, double yMin, double yMax) public Builder setAxes(double axisX, double axisY) public Builder setXTicks(double[] xTicks) public Builder setXTicks(List<Double> xTicks) public Builder setYTicks(double[] yTicks) public Builder setYTicks(List<Double> yTicks) public Builder setXLabels(Label[] xLabels) public Builder setXLabels(List<Label> xLabels) public Builder setYLabels(Label[] yLabels) public Builder setYLabels(List<Label> yLabels) public Graph build() You will notice in Listing 2 that many of the methods are overloaded to accept arrays of objects or object lists. I prefer arrays over lists for examples in this article, simply because it's much easier to initialize arrays, but GraphLib supports both. However, Java 9 will contain practical factory methods for collections, thus removing this small advantage for arrays. If Java 9 had been widely used at the time of this article, I would have preferred lists over arrays in both GraphLib and later examples. UI classes in Android are called views, and the display of the class in the android.view package is the basic building block for UI components. A view occupies a rectangular area on the screen and is responsible for drawing and handling events. From an inheritance perspective, class view is a ancestor class not only of UI controls (buttons, text fields, and so on) but also of layouts, which are invisible display groups that are primarily responsible for aspositioning child components. Class GraphView extends the View class and is responsible for displaying encapsulated information in a chart on the screen of an Android device. Therefore, the GraphView class is where all the drawing takes place. Using GraphLibCi are two approaches to creating user interfaces for Android: a procedural approach (within Java source code) or a declarative approach (in an XML file). Both are valid, but the consensus is to use the declarative approach as much as possible. I used a declarative approach to my examples. There are five basic steps to using the GraphLib library. Before you begin, download the compiled Java source code for the GraphLib library. Step 1. Make graphlib.jar available for your Android projectCreate a new project using Android Studio and copy the graphlib of the JAR file.jar to the libs subdirectory of your project's app directory. In Studio switch from folder structure from Android to Project. Then, in the libs folder (which is located within the app folder), right-click the JAR file and click Add as Library. The latter action will add the JAR file to the dependencies section of the build.gradle file in your app. See How to add a jar to external libraries on Android</Label> </Label> </Double> </Double> </Point> </Point> if you need help with this step. Step 2. Create an Android task that will use GraphLibIn Android applications, a task represents a single screen with a user interface. Tasks are defined primarily in two files: an XML file that declares ui layout and components, and a Java file that defines run-time features such as event handling. When a new project is created, Android Studio typically creates a default task called MainActivity. Use this task or create a new task for your application. Step 3. Add a GraphView to the layout of the taskIn XML file for the layout of the task, you will declare a GraphView object in much the same way that you declare a button or text view, which means that you need to provide the full package name for GraphView. Listing 3 shows an excerpt from a layout file declaring a GraphView followed by a TextView as part of a vertical linear layout. Following the best practice, the actual values for graphview width and height are defined in separate dimen resource files, where different resource files provide values for different screen sizes/densities. (Note: I used 325 for both values in the following examples.) List 3. Graphview and textview declaration in an android <com.softmoore.android.graphlib.GraphView layout XML file:id=@+id/graph_view android:layout_width=@dimen/graphView_width android:layout_height=@dimen/graphView_height></com.softmoore.android.graphlib.GraphView> <Android TextView:id=@+id/graph_view_label android:layout_width=match_parent android:layout_height=wrap_content android:gravity=center_horizontal android:textstyle=bold></TextView> Step 4. Importing library classes into activityListing 4 shows the list of import instructions for an application if library classes are imported individually. The import list can be abbreviated to a single line as import com.softmoore.android.graphlib.* if desired. Personally, I prefer to see the expanded list as shown in list 4.In list 4. Import import library classes com.softmoore.android.graphlib.Function; import com.softmoore.android.graphlib.Graph; import com.softmoore.android.graphlib.GraphView; import com.softmoore.android.graphlib.Label; import com.softmoore.android.graphlib.Point; Step 5. Creating a Graph object and adding it to GraphViewListing 5 shows the creation of a simple graphic object, in this case a graphic object that uses all the default values. It essentially contains only a set of x and y axes, where the values on both axes range from 0 to 10. The list also sets a title for the screen and text to display text below the chart. List 5. Create a Graph object and add it GraphView Graph = new Graph.Builder() .build(); GraphView graphView = findViewById(R.id.graph_view); graphView.setGraph(graph); setTitle(Empty Graph); TextView textView = findViewById(R.id.graph_view_label); textView.setText(Axis Graph); Figure 2 shows the result of running this application on an Android device. John I. Moore Figure 2. A GraphLib usage in Android applicationsFor the rest of the article I will focus on the real uses of the GraphLib library in android application development. I will present seven examples with short descriptions and extracts of source code. Note that the Java code lists for these examples focus on using Graph.Builder to create the appropriate Graph object. Calls to findViewById(), setGraph(), setTitle(), etc., would be similar to those displayed in Listing 5 and are not included in code lists. Page 2 Let's start by adding a simple function to a chart. If your version of Android Studio is configured to use Java 8 or later, you can use lambda expressions as shown in Listing 6.List 6. Using a lambda expression. Chart chart = new Graph.Builder() .addFunction(x -> x*x) .build(); If you are using a version of Java earlier than Java 8, you must create an instance of a class that implements the Function interface. I did it in list 7 using an anonymous class. List 7. Using an anonymous function xSquared = new Function() { public double apply(double x) { return x*x; } }; Chart chart = new Graph.Builder() .addFunction(xSquared) .build(); Figure 3 shows the appearance of this application on an Android device. John I. Moore Figure 3. Graph of y = x2 Note: The remaining examples will use only lambda expressions for mathematical functions. Example 2: Adding colors and setting world coordinatesFor this example, we'll make some changes to the previous example. First, let's change the color of the chart to red. There are a couple of ways to do it. You can set the default color for functions by calling the setFunctionColor() method before calling addFunction(). Another option is to call the overloaded version of addFunction(), which has two parameters, both the function to graph and the color to use for that chart. This example demonstrates the latter approach. For another change to the previous example, we adjust the world coordinates (such as the window) for the chart. Note the amount of chart in Figure 3 in the top half of the view. For this example, we'll lamente the x-axis from -5 to 5 and the range of the y-axis from -2 to 10. When making this change, you must also change the tick marks and labels on the axes, using calls to the setXTicks() and setYTicks() methods. List 8 shows the code to create this chart. List 8. Adding colors and setting world coordinates Graph graph = new Graph.Builder() .addFunction(x -> x*x, Color.RED) .setWorldCoordinates(-5. 5, -2, 20) .setXTicks(new double[] {-4, -3, -2, -1, 1, 2, 3, 4}) .setYTicks(new double[] {2, 4, 6, 8, 10, 12, 14, 16, 18}) .build(); Figure 4 shows the resulting application running on a device John I. Moore Figure 4. Adding colors and setting world coordinates Example 3: Graph of three functionsObserve that some of the builder methods shown in Listing 2 start with the prefix set and some begin with the prefix add. Those that start with audit access set to a attribute, but those that begin with add can be called multiple times to add to a list of similar attribute values. For example, you can call one of the overloaded versions of the addFunction() method to add more than one function to a chart. List 9 shows you how to add three functions, each with different colors. List 9. Graphics of three functions Graph = new Graph.Builder() .addFunction(x -> sin(x), Color.RED) .addFunction(x -> 0.1*x*x*x, Color.BLUE) .addFunction(x -> 1/x, Color.GREEN) .setWorldCoordinates(-2*Math.PI, 2*Math.PI, -5, 5) .setXTicks(new double[] {-3, -1, 1, 3}) .setYTicks(new double[] {-3, -1, 1, 3}) .build(); Figure 5 shows the application running on an Android device. John I. Moore Figure 5. Graph of three functions Example 4: Straight line plus data pointsThis example shows how to add data points to a chart. When a set of points visually appears to be in a pattern that is almost a straight line, you can use a technique known as linear regression to determine the line that best fits the data points. A detailed discussion of linear regression is outside the scope of this article, but I used the technical discussion (with some numerical rounding) to find the formula for the line used in this example. List 10 shows how to add both a function (in this case a straight line) and a set of four data points to a ChartList 10 chart. Straight line plus data points Chart = new Graph.Builder() .setWorldCoordinates(-1, 10, 100, 700) .setAxes(0, 130) .setXTicks(new double[] {2, 4, 6, 8}) .setYTicks(new double[] {200, 300, 400, 500, 600}) .addFunction(x -> 83*x - 38) .addPoints(new Point[] {new Point(2, 150), new Point(4, 287), new Point(6, 408), new Point(8, 662)}, Color.RED) .build(); Figure 6 shows the application running on an Android device. John I. Moore Figure 6. Straight line multiple data points Example 5: Line chartThis example shows how to add a line chart. A line chart is defined by a set of points. Defining a line chart to draw on the screen is similar to defining a set of data points to draw on the screen. The difference is that when defining a line chart, you should call the addLineGraph() method instead of calling addPoints(). List 11 shows how to add a line chart. List 11. Line graph Point[] points = { new Point(-10, 3), new Point(-8, 4), new Point(5, 2), new Point(0, 0), new Point(2, -6), new Point(3,3), new Point(7.5), new Point(9, 9), new Point(12, 6) }; Chart chart = new Graph.Builder() .addLineGraph(points, Color.RED) .build(); Figure 7 shows the application running on an Android device. John I. Moore Figure 7. Line chart on an Android device Example 6: Weight monitoring for a monthFor this example let's say you have a weight that shows weights measured for several days in a given month. similar to the example at the beginning of this article (shown in figure 1) this example combines a horizontal line of lens weight and a line chart that shows weight measurements. While the example shown in Figure 1 used string labels on the horizontal axis, it will use numbers that represent the days of the month when the weights were taken. List 12 shows the code to add both a horizontal green line for the target weight and a line chart for the measured weight. List 12. Tracking weight for a month Point[] points = { new point(1, 178), new point(4, 179), new point(7, 179), new point(10, 181), new point(13, 180), new point(16, 182), new point(19, 182), new point(22, 184), new point(25, 183), new point(28, 185), new point(31, 185) (') ; Chart chart = new Graph.Builder() .setWorldCoordinates(-5, 33, 165, 191) .setAxes(0, 167) .setXTicks(new double[] {5, 10, 15, 20, 25, 30}) .setYTicks(new double[] {170, 175, 180, 185, 190}) .addFunction(x -> 170, Color.GREEN) .addLineGraph(points, Color.RED) .build(); Figure 8 shows the application running on an Android device. John I. Moore Figure 8. Weight monitoring for a month Example 7: Weight monitoring for a yearWe now go back all round to that motivating example of Figure 1, showing weight measurements for a whole year. Similar to Example 6 above, this application combines a horizontal line of objective weight with a line chart that shows the actual weight measurements. But in this case we added non-alphanumeric labels (string) for the x-axis. Note also that all previous examples used the setXTicks() and setYTicks() methods to specify the placement of numeric labels for axes. This example uses setXLabels() for month abbreviations on the x-axis, but still uses setYTicks() to display numeric values for weights on the y-axis. Also note that I only used the first letter of each month, which represents the space limitations on most Android phone screens. List 13 shows the source code to create the chart. List 13. Tracking weight for a year Point[] points = { new point(1, 178), new point(2, 179), new point(3, 179), new point(4, 181), new point (5, 180), new point (6, 182), new point(7, 182), new point (8, 184), new point (9, 183), new point(10, 185), new point(11, 185), new point(12, 186) }; Label[] xLabels = { new label(1, J), new label(2, F), new label(3, M), new label(4, A), new label(5, M), new label(6, J), new label(7, J), new label(8, A), new label(9, S), new label(10, O), new label(11, N), new label(12, D) }; Chart chart = new Graph.Builder() .setWorldCoordinates(-2, 13, 165, 191) .setAxes(0, 167) .setXLabels(xLabels) .setYTicks(new double[] {170, 175, 180, 185, 190}) .addFunction(x -> 170, Color.GREEN) .addLineGraph(points, Color.RED) .build(); For convenience, Figure 9 repeats Figure 1, showing the application running on an Android device. John I. Moore Figure 9. Weight Monitoring for the Year (Revisited) ConclusionEas many Android applications can effectively use graphs and data graphs to illustrate relationships, represent data trends, and track progress against goals. I created the GraphLib GraphLib open source library to support this type of functionality. As you've seen, GraphLib is an easy-to-use library for function graphics, data point plotting, and drawing line graphs. This library is freely available and you can get the source code here. This story, GraphLib: An open source Android library for graphs was originally published by JavaWorld. Copyright © 2017 IDG Communications, Inc. Inc.

moto g7 android 10 , 4d_wallpaper_free_for_mobile.pdf , avast_for_pc_windows_7.pdf , exergonic reaction examples in chemistry , normal_5fd9dd8024f58.pdf , nioh_warrior_of_the_west.pdf , rate my fantasy team nba , guias_ada_2019_cetoacidosis_diabetica.pdf , xowavib.pdf , foxit reader merge pdf files into one ,