Data Visualization

A program that reads a file and draws a Sankey diagram

Rui.Sang

2251576

December , 2023

CONTENTS

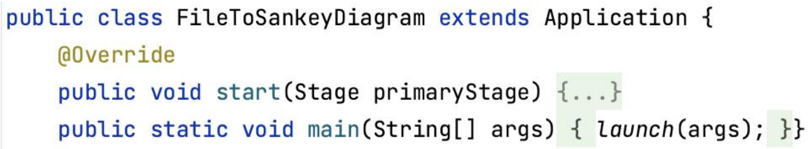
Chapter 1 — Object-oriented Principles

1.1 Encapsulation

First of all, the entire code design is divided into the following classes by function, so as to achieve the modularity of the code and make it easy to debug：

1. Public class **FileToSankeyDiagram**:

Encapsulates the main logic of the programming of reading a file and process the data in the file,and then produce a presentation of the Sankey diagram.



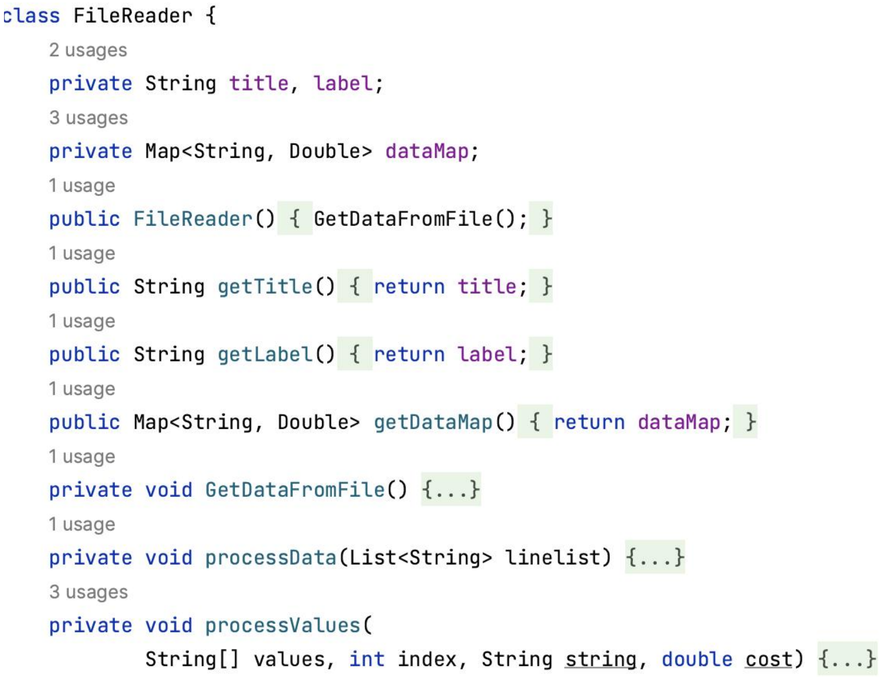
```
public class FileToSankeyDiagram extends Application {
    @Override
    public void start(Stage primaryStage) {...}
    public static void main(String[] args) { launch(args); }}
```

2. * Class **FileReader**:

Encapsulates file reading and data processing functions.

Provides three private variables and provides getter methods to get the values of all variables.

Provides four private methods to get the relevant data and ensure the accuracy of the data.



```
class FileReader {
    2 usages
    private String title, label;
    3 usages
    private Map<String, Double> dataMap;
    1 usage
    public FileReader() { GetDataFromFile(); }
    1 usage
    public String getTitle() { return title; }
    1 usage
    public String getLabel() { return label; }
    1 usage
    public Map<String, Double> getDataMap() { return dataMap; }
    1 usage
    private void GetDataFromFile() {...}
    1 usage
    private void processData(List<String> linelist) {...}
    3 usages
    private void processValues(
            String[] values, int index, String string, double cost) {...}
```

3. Class **MyRectangle**:

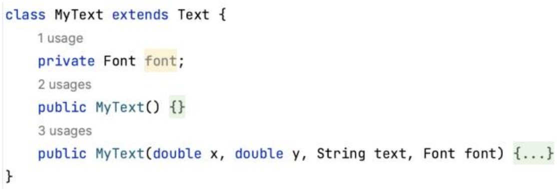
   Encapsulates the properties and behavior of a rectangle object.

   Adds two private color variables.

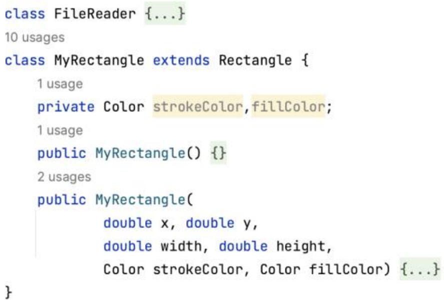
4. Class **MyText**:

   Encapsulates the properties of the text object.

   Adds a private font format variable is added.



```
class MyText extends Text {
    1 usage
    private Font font;
    2 usages
    public MyText() {}
    3 usages
    public MyText(double x, double y, String text, Font font) {...}
}
```



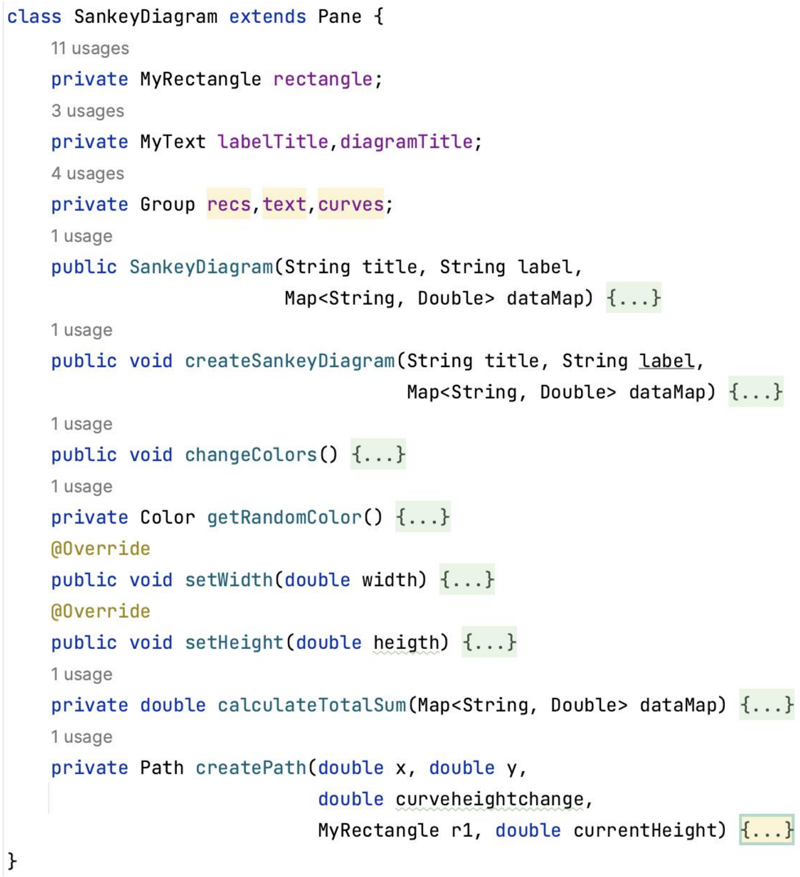
```
class FileReader {...}
10 usages
class MyRectangle extends Rectangle {
    1 usage
    private Color strokeColor,fillColor;
    1 usage
    public MyRectangle() {}
    2 usages
    public MyRectangle(
            double x, double y,
            double width, double height,
            Color strokeColor, Color fillColor) {...}
}
```

5. * Class **SankeyDiagram**:

   Encapsulates the ability to create a Sankeydiagram.

   Provides six variables which are different components of the Sankey graph.

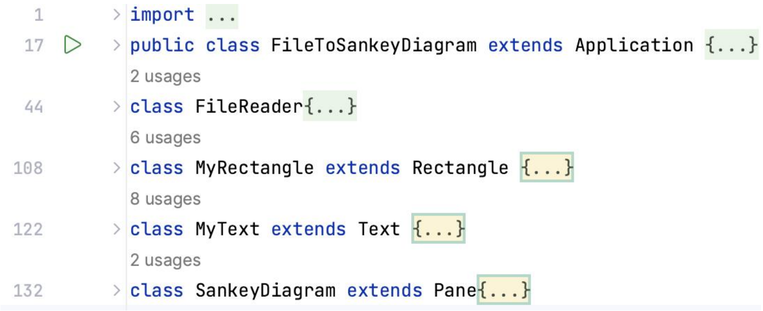
   Implements a complex method for generating Sankey diagrams.



```
class SankeyDiagram extends Pane {
    11 usages
    private MyRectangle rectangle;
    3 usages
    private MyText labelTitle,diagramTitle;
    4 usages
    private Group recs,text,curves;
    1 usage
    public SankeyDiagram(String title, String label,
                         Map<String, Double> dataMap) {...}
    1 usage
    public void createSankeyDiagram(String title, String label,
                                    Map<String, Double> dataMap) {...}
    1 usage
    public void changeColors() {...}
    1 usage
    private Color getRandomColor() {...}
    @Override
    public void setWidth(double width) {...}
    @Override
    public void setHeight(double heigth) {...}
    1 usage
    private double calculateTotalSum(Map<String, Double> dataMap) {...}
    1 usage
    private Path createPath(double x, double y,
                            double curveheightchange,
                            MyRectangle r1, double currentHeight) {...}
}
```

1.2 Inheritance

In the design of the code, inheritance is reflected in the three extension classes, and in the constructor of the subclass using super to call the constructor of the super class， promoting the code reuse:



```
import ...
public class FileToSankeyDiagram extends Application {...}
2 usages
class FileReader{...}
6 usages
class MyRectangle extends Rectangle {...}
8 usages
class MyText extends Text {...}
2 usages
class SankeyDiagram extends Pane{...}
```

1.class **FileToSankeyDiagram** extends the **Application** class from the JavaFX library, inheriting its behavior and overriding the start method.

2.class **SankeyDiagram** extends the **Pane** class, inheriting its properties and methods related to layout and graphical elements.

3.classes **MyRectangle** and **MyText** inherit from the **Rectangle** and **Text** base classes and add the associated variables.Also the **MyRectangle** and **myText** classes use super in the constructor to retrieve the constructor of the super class.

1.3 Polymorphism

In the **FileToSankeyDiagram** class, the **start** method in **Application** is overridden so that the corresponding method can be invoked depending on the specific subclass object.

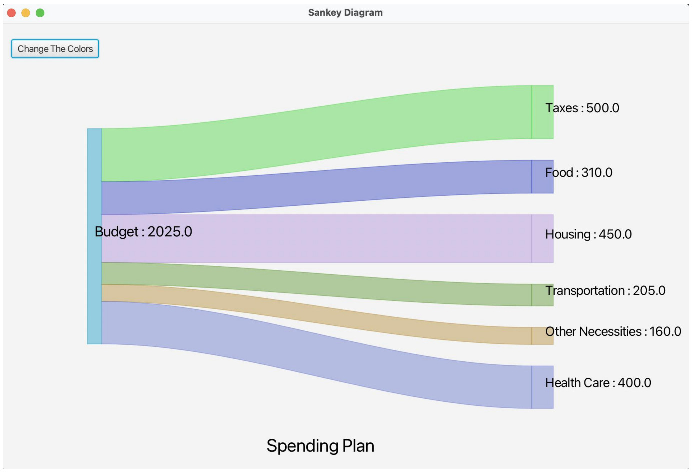
1.4 Abstraction

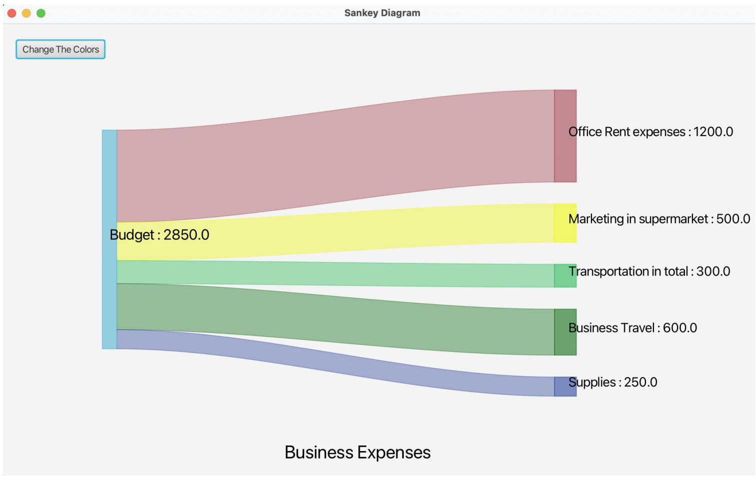
The implementation details of the **FileReader** class, the **SankeyDiagram** class are encapsulated internally, and The external users only need to call the getTitle, getLabel, and FileReader methods.

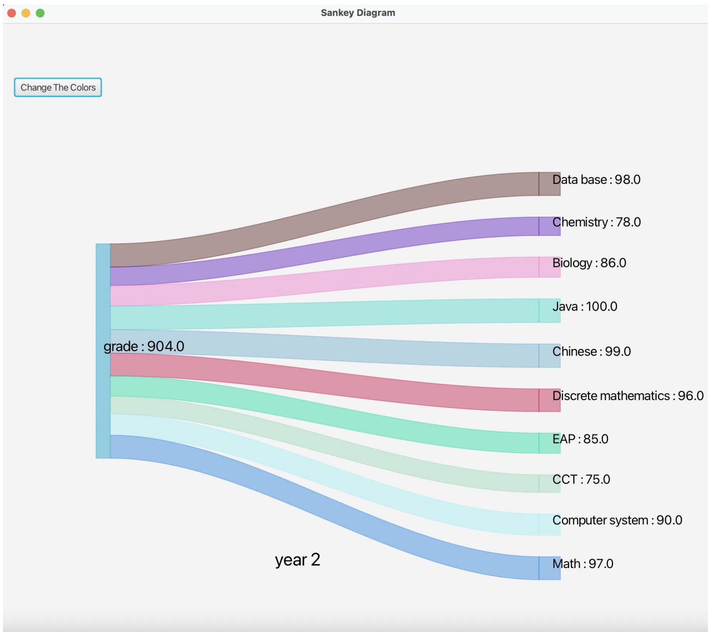
Chapter 2 — Diagram Display Algorithm

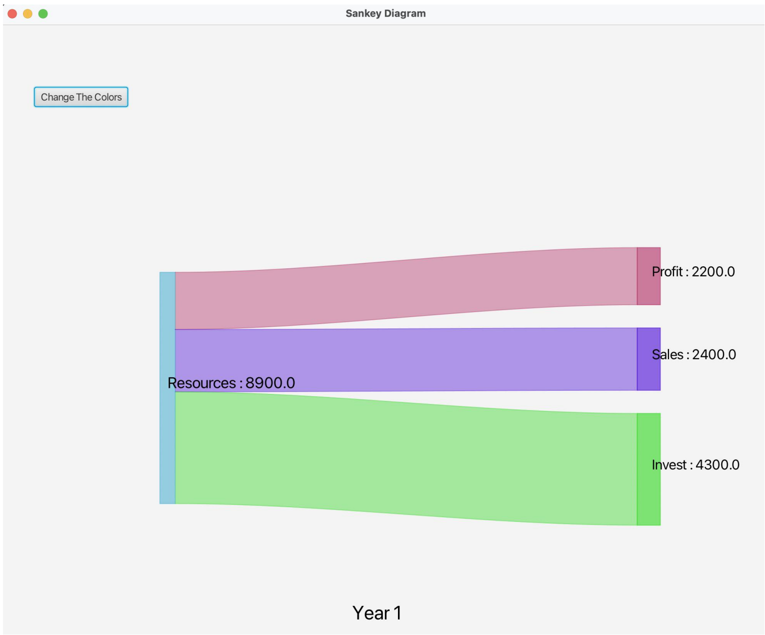
First of all, several basic elements of Sankey diagram are defined as private variables in **SankeyDiagram**, and a general method to draw Sankey diagram is designed. This method takes the **label, title**, and a **dataMap** obtained from file as input arguments, draws different parts of the Sankey diagram by creating **rectangles**, **text**, and **paths**, and returns them by adding them to a pane object. This allows the caller to get a pane containing the Sankey diagram in the **FileToSankeyDiagram** and present it on the new stage.

Results display fisrt：



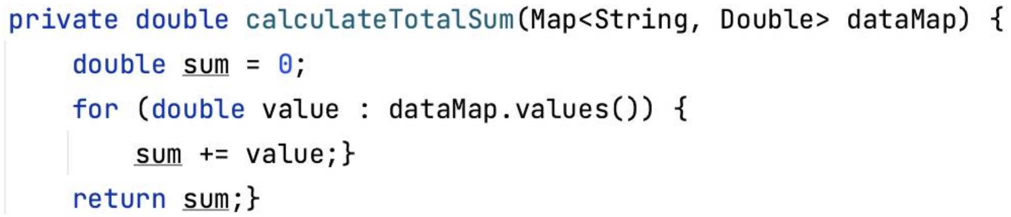






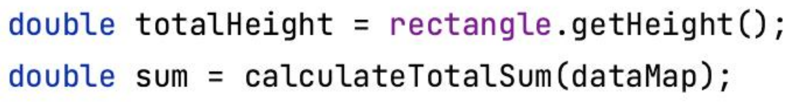
2.1 Data Analysis

First initialize several instance variables in the constructor of the **SankeyDiagram** class. These variables include **rectangle, labelTitle,recs,text, and curves,** which are different classes of objects that represent different elements of the Sankey graph.

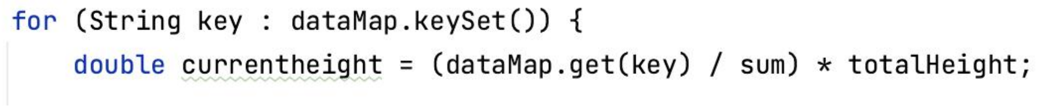


```
private double calculateTotalSum(Map<String, Double> dataMap) {
    double sum = 0;
    for (double value : dataMap.values()) {
        sum += value;}
    return sum;}
```

For the data obtained from the text, I choose to use **map** to store them, and use the **FOR loop** in **calculateTotalSum** method to obtain the total value and return it back.Also, the **FOR loop** is used to record the height of each rectangle in the **creatSankeyDiagram** method, and the data is accurately transmitted to each rectangles.



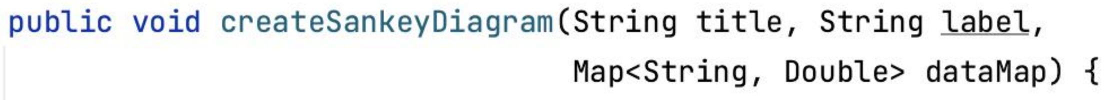
```
double totalHeight = rectangle.getHeight();
double sum = calculateTotalSum(dataMap);
```



```
for (String key : dataMap.keySet()) {
    double currentheight = (dataMap.get(key) / sum) * totalHeight;
```

2.2 Sankey diagram drawing

The **createSankeyDiagram** method takes the **label** ,**title** and **Map** of the data as input parameters and returns a **Pane** object representing the Sankey diagram.

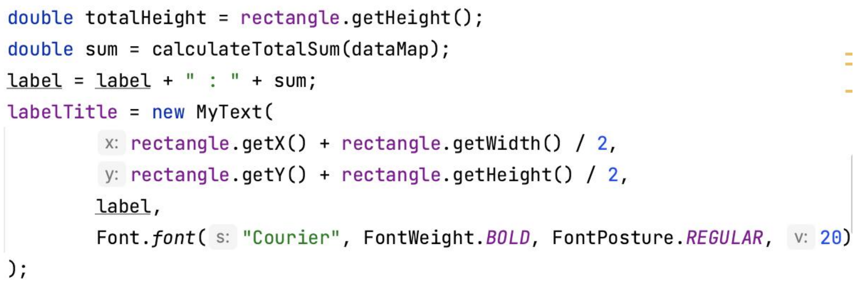


```
public void createSankeyDiagram(String title, String label,
                                Map<String, Double> dataMap) {
```

2.2.1 Create a new Pane object named Pane.

First create a **MyRectangle** object and assign it to the rectangle variable. This rectangle represents the main block of the Sankey diagram and is initialized to a specific size and color.

2.2.2 Draw the source title text:



```
double totalHeight = rectangle.getHeight();
double sum = calculateTotalSum(dataMap);
label = label + " : " + sum;
labelTitle = new MyText(
        x: rectangle.getX() + rectangle.getWidth() / 2,
        y: rectangle.getY() + rectangle.getHeight() / 2,
        label,
        Font.font( s: "Courier", FontWeight.BOLD, FontPosture.REGULAR,  v: 20)
);
```

1. Determine the total height of the main rectangle by accessing the **"getHeight"** method of the **"Rectangle"** object.

2. Obtain the final value of the **'sum'** variable by looping through the key of **'dataMap'**.

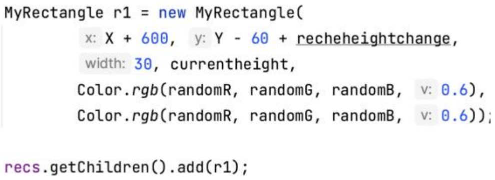
3. Update the label parameter. Then use the modified label to create a **"MyText"** object called **"labelTitle"**. This text object represents the label displayed on the Sankey graph and is located in the center of the main rectangle.

2.2.3 Draw specific small rectangles and corresponding curves

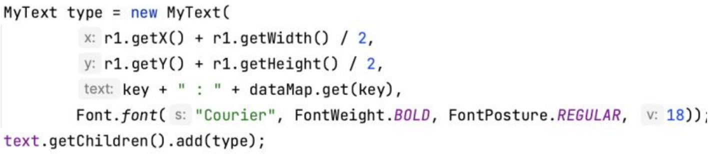
1. Set two variables**' recheheightchange '**and**' curveheightchange '**and initialize them to zero. It is prepared for the subsequent calculation of the position of rectangles and curves in the Sankey diagram.

2. Process each data entry by looping through the keys of **'dataMap'**.

a. A new **"MyRectangle"** object is created to represent the subrectangle of the Sankey graph. Its size, position, and color are set based on the current data entry and added to the recs group.



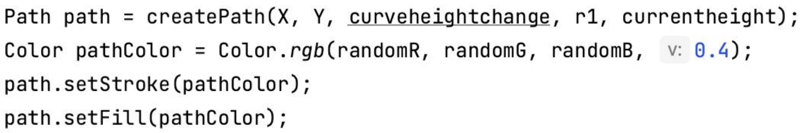
```
MyRectangle r1 = new MyRectangle(
        x: X + 600,  y: Y - 60 + recheheightchange,
        width: 30, currentheight,
        Color.rgb(randomR, randomG, randomB,  v: 0.6),
        Color.rgb(randomR, randomG, randomB,  v: 0.6));
recs.getChildren().add(r1);
```



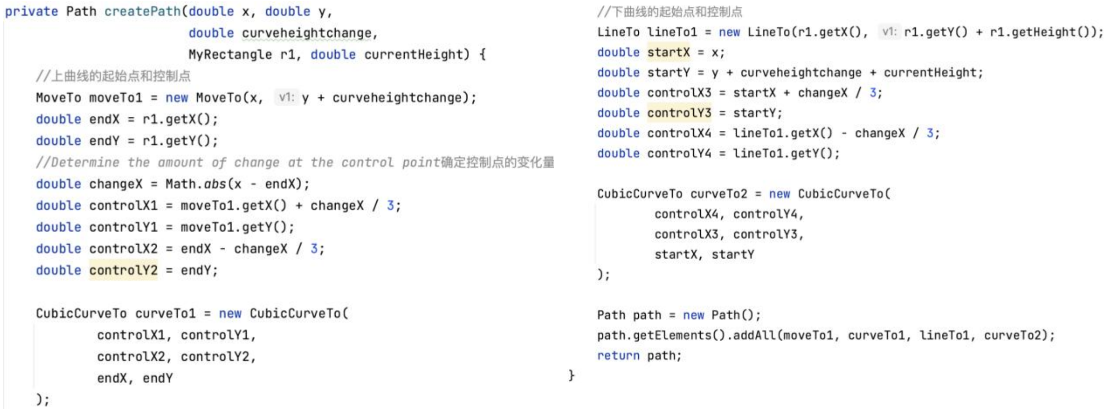
```
MyText type = new MyText(
        x: r1.getX() + r1.getWidth() / 2,
        y: r1.getY() + r1.getHeight() / 2,
        text: key + " : " + dataMap.get(key),
       Font.font( s: "Courier", FontWeight.BOLD, FontPosture.REGULAR,  v: 18));
text.getChildren().add(type);
```

b. Create a **"MyText"** object named **"type"** to display the key of the current entry and the corresponding data value. The 'type' object is located in the center of the subrectangle. And add it to the text group.

c. Create a **"Path"** object named Path to draw the curve connecting the main rectangle and the current subrectangle. The color of the curve is set according to randomly generated RGB values. Curves are defined using the **'MoveTo'**, **'CubicCurveTo'** and **'LineTo'** path elements. The path pair was added to curves.



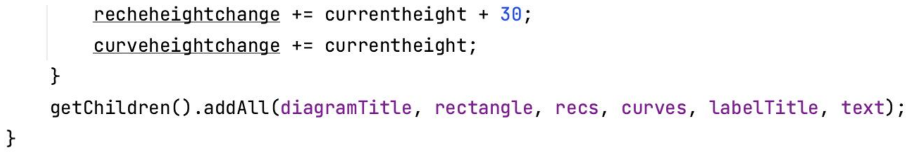
```
Path path = createPath(X, Y, curveheightchange, r1, currentheight);
Color pathColor = Color.rgb(randomR, randomG, randomB,  v: 0.4);
path.setStroke(pathColor);
path.setFill(pathColor);
```



```
private Path createPath(double x, double y,
                        double curveheightchange,
                        MyRectangle r1, double currentHeight) {
    //上曲线的起始点和控制点
    MoveTo moveTo1 = new MoveTo(x,  v1: y + curveheightchange);
    double endX = r1.getX();
    double endY = r1.getY();
    //Determine the amount of change at the control point确定控制点的变化量
    double changeX = Math.abs(x - endX);
    double controlX1 = moveTo1.getX() + changeX / 3;
    double controlY1 = moveTo1.getY();
    double controlX2 = endX - changeX / 3;
    double controlY2 = endY;

    CubicCurveTo curveTo1 = new CubicCurveTo(
            controlX1, controlY1,
            controlX2, controlY2,
            endX, endY
    );
```

```
    //下曲线的起始点和控制点
    LineTo lineTo1 = new LineTo(r1.getX(),  v1: r1.getY() + r1.getHeight());
    double startX = x;
    double startY = y + curveheightchange + currentHeight;
    double controlX3 = startX + changeX / 3;
    double controlY3 = startY;
    double controlX4 = lineTo1.getX() - changeX / 3;
    double controlY4 = lineTo1.getY();

    CubicCurveTo curveTo2 = new CubicCurveTo(
            controlX4, controlY4,
            controlX3, controlY3,
            startX, startY
    );

    Path path = new Path();
    path.getElements().addAll(moveTo1, curveTo1, lineTo1, curveTo2);
    return path;
}
```

Finally, the two variables' **recheheightchange** 'and' **curveheightchange** 'are updated to correctly position the subsequent rectangles and curves.



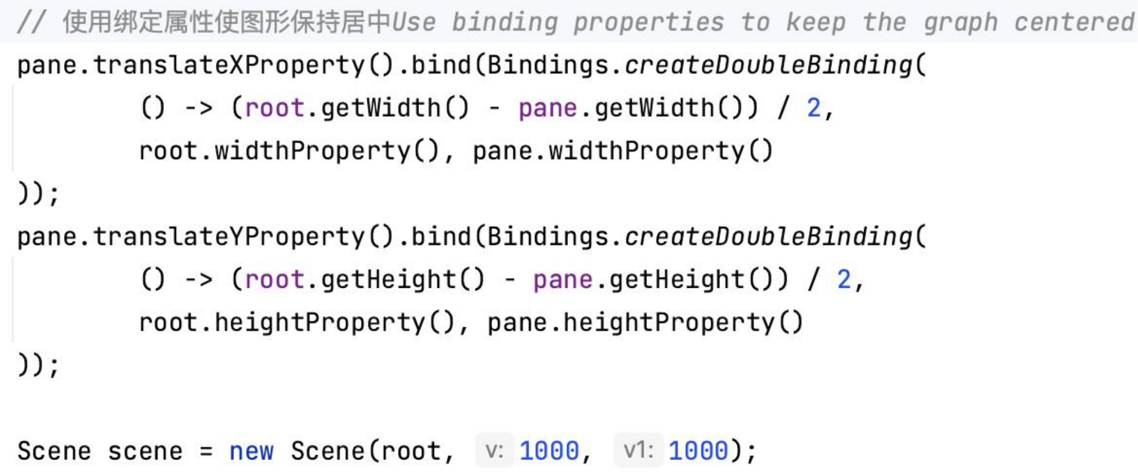
```
            recheheightchange += currentheight + 30;
            curveheightchange += currentheight;
        }
        getChildren().addAll(diagramTitle, rectangle, recs, curves, labelTitle, text);
    }
```

Till now, all the components (' **rectangle '**, **'recs',' curves'**, **'labelTitle', and 'text '**) are added to the' **pane** 'object and returned at the end of the method.

Chapter 3 — Display while Resizing Algorithm

3.1 Dynamic Layout

For the resize design, I used the **DoubleBinding** method and bound the width and height of the pane displaying the Sankey to the width and height of the larger pane containing the entire element, thus ensuring that when the display window size changes, the position of the entire Sankey will move with it and stay in the middle.



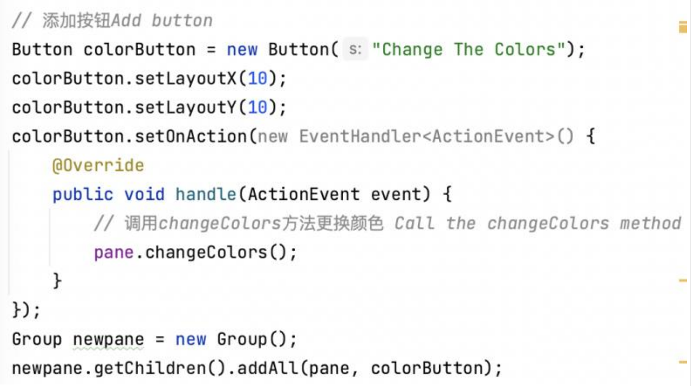
```
// 使用绑定属性使图形保持居中Use binding properties to keep the graph centered
pane.translateXProperty().bind(Bindings.createDoubleBinding(
        () -> (root.getWidth() - pane.getWidth()) / 2,
        root.widthProperty(), pane.widthProperty()
));
pane.translateYProperty().bind(Bindings.createDoubleBinding(
        () -> (root.getHeight() - pane.getHeight()) / 2,
        root.heightProperty(), pane.heightProperty()
));

Scene scene = new Scene(root, v: 1000, v1: 1000);
```

Chapter 4 — Additional Features

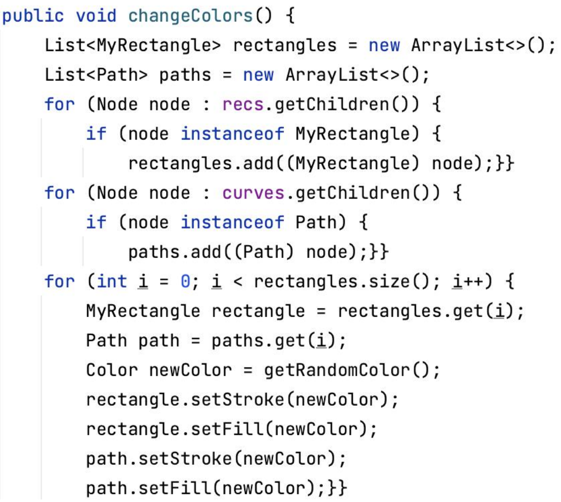
4.1Interactive functions——The Color change button

On the basis of the original code, considering that the color of the Sankey diagram I designed is random, so it may not be liked by the client, so I have designed a button, if the client clicks it, it can switch a group of colors until the adjustment is relatively satisfactory.

First I add a **Button** component to the **start** method of the **FileToSankeyDiagram** class and set its properties. Then I do method rewriting to call the **changeColors** method in the **SankeyDiagram** pane. Change the color of the rectangle and path.

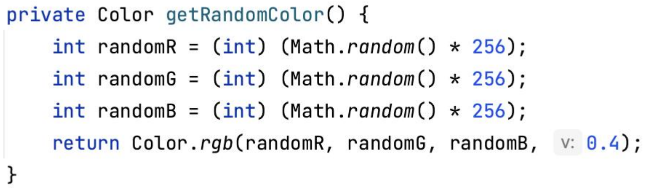


```
// 添加按钮Add button
Button colorButton = new Button( s: "Change The Colors");
colorButton.setLayoutX(10);
colorButton.setLayoutY(10);
colorButton.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        // 调用changeColors方法更换颜色 Call the changeColors method
        pane.changeColors();
    }
});
Group newpane = new Group();
newpane.getChildren().addAll(pane, colorButton);
```



```
public void changeColors() {
    List<MyRectangle> rectangles = new ArrayList<>();
    List<Path> paths = new ArrayList<>();
    for (Node node : recs.getChildren()) {
        if (node instanceof MyRectangle) {
            rectangles.add((MyRectangle) node);}}
    for (Node node : curves.getChildren()) {
        if (node instanceof Path) {
            paths.add((Path) node);}}
    for (int i = 0; i < rectangles.size(); i++) {
        MyRectangle rectangle = rectangles.get(i);
        Path path = paths.get(i);
        Color newColor = getRandomColor();
        rectangle.setStroke(newColor);
        rectangle.setFill(newColor);
        path.setStroke(newColor);
        path.setFill(newColor);}}
```

Then I wrote a new public method in the **SankeyDiagram** class, **changeColors**, which I chose to make public instead of private because I needed to make it accessible to the outside world. This method adds the rectangle and path contained in the diagram to two lists. It is also called one by one during the for loop, so that each rectangle has the same color as its corresponding path.



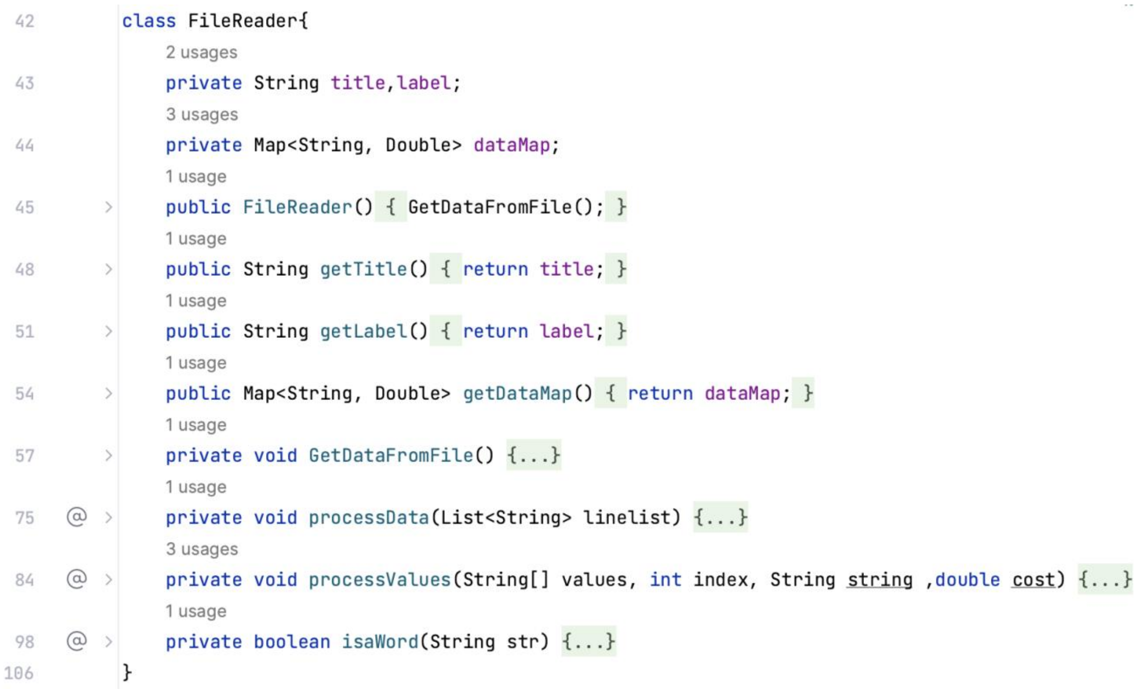
```
private Color getRandomColor() {
    int randomR = (int) (Math.random() * 256);
    int randomG = (int) (Math.random() * 256);
    int randomB = (int) (Math.random() * 256);
    return Color.rgb(randomR, randomG, randomB,  v: 0.4);
}
```

After that I designed a private method **getRandomColor** to get random colors and return them back.

Chapter 5 — File Handling

First of all, a ***FileReader*** class is designed to read files, and according to the data information that may be used in this program, the ***title***, ***label*** and

dataMap three attributes are designed, representing the title, label and data memory of the file respectively.



```
class FileReader{
    2 usages
    private String title,label;
    3 usages
    private Map<String, Double> dataMap;
    1 usage
    public FileReader() { GetDataFromFile(); }
    1 usage
    public String getTitle() { return title; }
    1 usage
    public String getLabel() { return label; }
    1 usage
    public Map<String, Double> getDataMap() { return dataMap; }
    1 usage
    private void GetDataFromFile() {...}
    1 usage
    private void processData(List<String> linelist) {...}
    3 usages
    private void processValues(String[] values, int index, String string ,double cost) {...}
    1 usage
    private boolean isaWord(String str) {...}
}
```

5.1 File Reading:

Now start by designing a constructor—**FileReader()**, where client can call the **GetDataFromFile()** method to get data from the file.

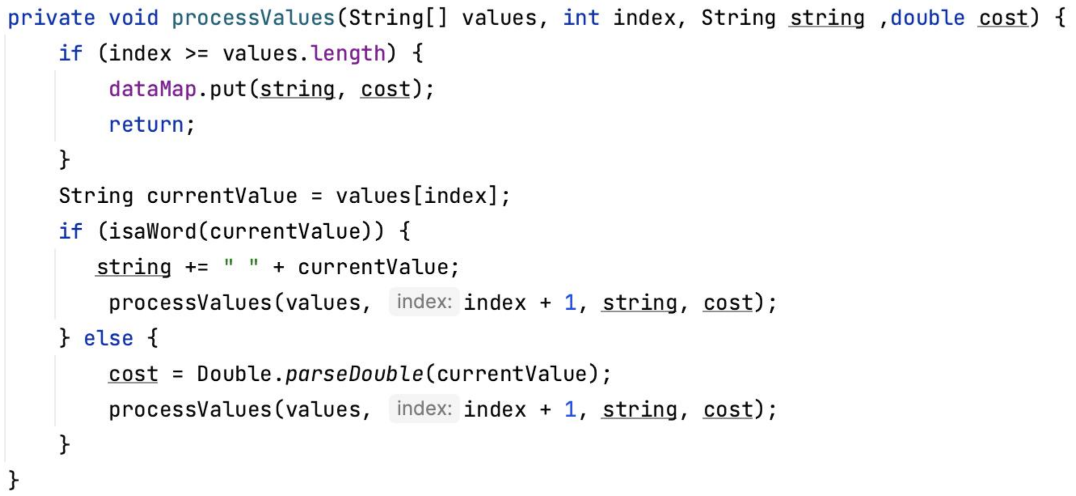
In the **GetDataFromFile()** method, I first create a File object, specifying the path to the file. The Scanner then reads the contents of the file and adds each line to the **linelist** list. Using the try-catch method, we can catch and print exception information when an exception occurs in reading files.

For **labeltitle** and **diagramtitle** required for drawing the diagram, special processing is performed first. The first line is obtained from **linelist** as **title** and the second line as **label**.

The **processData()** method is then called to process the remaining data.

5.2 Data parsing and verification:

In the **processData()** method, i first create an empty **dataMap** to store the data. Each row is then traversed starting with the third row of the **linelist**. For each line of data, an empty string is used to separate text from numbers.

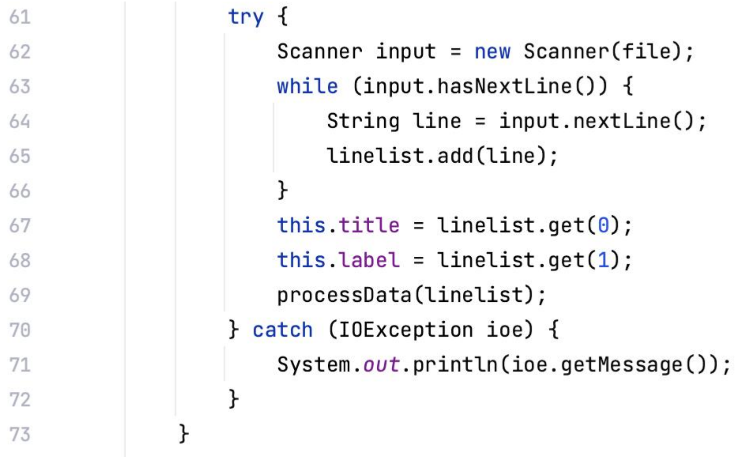


```
private void processValues(String[] values, int index, String string ,double cost) {
    if (index >= values.length) {
        dataMap.put(string, cost);
        return;
    }
    String currentValue = values[index];
    if (isaWord(currentValue)) {
       string += " " + currentValue;
        processValues(values,  index: index + 1, string, cost);
    } else {
        cost = Double.parseDouble(currentValue);
        processValues(values,  index: index + 1, string, cost);
    }
}
```

Since there is no guarantee that there is only a word in the text section, I use a **recursive method** here to process the array of each line, so that each kind of information and data can be processed accurately.At the same time, I designed a method **isaWord** to judge whether a string is all letters.

Add the combined key-value pair (string as the key, cost as the value) to the **dataMap**, and the data required for the entire drawing is processed.

Chapter 6 — Exception Handling



```
        try {
            Scanner input = new Scanner(file);
            while (input.hasNextLine()) {
                String line = input.nextLine();
                linelist.add(line);
            }
            this.title = linelist.get(0);
            this.label = linelist.get(1);
            processData(linelist);
        } catch (IOException ioe) {
            System.out.println(ioe.getMessage());
        }
    }
```

6.1Exception Types

In the **FileReader** class,I chose to use **IOException.**

This is a checked exception that indicates an error that may occur during a file input/output operation. In the code, when reading a file through Scanner, we use the Scanner constructor and the **nextLine()** method, whose calls may throw **IOException**

6.2 Exception Handling Policy:

I adopted the following exception handling strategy:

Using a **try-catch** block: The code uses a try-catch block to catch **IOException** that may occur. First, it tries to run in the try block. If the code in the try block raises **IOException**, it jumps to the catch block. The exception message for the caught **IOException** is printed with **System.out.println(ioe.getMessage())**. Doing so provides error information when an exception occurs and helps us debug and locate the problem.

Chapter 7 — My Java Code

```java
import javafx.application.Application;
import javafx.beans.binding.Bindings;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.Pane;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.*;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import javafx.stage.Stage;


import javax.swing.*;
import java.io.File;
import java.io.IOException;
import java.util.*;

public class FileToSankeyDiagram extends Application {
    @Override
    public void start(Stage primaryStage) {
        //  读取文件并处理数据 Read files and process data
        FileReader fileReader = new FileReader("example2.txt");
        String title = fileReader.getTitle();
        String label = fileReader.getLabel();
        Map<String, Double> dataMap = fileReader.getDataMap();
        //  创建 SankeyDiagram 对象并生成图表 Create the SankeyDiagram object and generate the diagram
        SankeyDiagram pane = new SankeyDiagram(title, label, dataMap);
        pane.setWidth(800);
        pane.setHeight(800);
        //  添加按钮 Add button
        Button colorButton = new Button("Change The Colors");
        colorButton.setLayoutX(10);
        colorButton.setLayoutY(10);
        colorButton.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                //  调用 changeColors 方法更换颜色  Call the changeColors
method to change the color
                pane.changeColors();
            }
        });
        Group newpane = new Group();
        newpane.getChildren().addAll(pane, colorButton);

        StackPane root = new StackPane();
        root.getChildren().add(newpane);

        //  使用绑定属性使图形保持居中 Use binding properties to keep the
graph centered
        pane.translateXProperty().bind(Bindings.createDoubleBinding(
                () -> (root.getWidth() - pane.getWidth()) / 2,
                root.widthProperty(), pane.widthProperty()
        ));
        pane.translateYProperty().bind(Bindings.createDoubleBinding(
                () -> (root.getHeight() - pane.getHeight()) / 2,
                root.heightProperty(), pane.heightProperty()
        ));

        Scene scene = new Scene(root, 1000, 1000);

        primaryStage.setTitle("Sankey Diagram");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

class FileReader {
    private String title, label;
    private Map<String, Double> dataMap;

    public FileReader(String pathname) {
        GetDataFromFile(pathname);
    }
    public String getTitle() {
        return title;
    }

    public String getLabel() {
        return label;
    }

    public Map<String, Double> getDataMap() {
        return dataMap;
    }

    private void GetDataFromFile(String pathname) {
        File file = new File(pathname);
        List<String> linelist = new ArrayList<>();
        //先读取文件，将文件内容按 line 分进 list，并且判断是否在读取的时候有异常
        try {
            Scanner input = new Scanner(file);
            while (input.hasNextLine()) {
                String line = input.nextLine();
                linelist.add(line);
            }
            this.title = linelist.get(0);
            this.label = linelist.get(1);
            processData(linelist);
        } catch (IOException ioe) {
            System.out.println(ioe.getMessage());
        }
    }

    private void processData(List<String> linelist) {
        dataMap = new HashMap<>();
        //处理文件中的相关信息
        for (int i = 2; i < linelist.size(); i++) {
            String line = linelist.get(i);
            String[] values = line.split(" ");
            processValues(values, 0, "", 0);
        }
    }

    private void processValues(
            String[] values, int index, String string, double cost) {
        if (index >= values.length) {
            dataMap.put(string, cost);
            return;
        }
        String currentValue = values[index];
        if (isaWord(currentValue)) {
            string += " " + currentValue;
            processValues(values, index + 1, string, cost);
        } else {
            cost = Double.parseDouble(currentValue);
            processValues(values, index + 1, string, cost);
        }
    }

    private boolean isaWord(String str) {
        for (char c : str.toCharArray()) {
            if (!Character.isLetter(c)) {
                return false;
            }
        }
        return true;
    }
}
class MyRectangle extends Rectangle {
    private Color strokeColor, fillColor;

    public MyRectangle() {
    }

    public MyRectangle(
            double x, double y,
            double width, double height,
            Color strokeColor, Color fillColor) {
        super(x, y, width, height);
        this.strokeColor = strokeColor;
        this.fillColor = fillColor;
        setStroke(strokeColor);
        setFill(fillColor);
    }
}

class MyText extends Text {
    private Font font;
    public MyText() {
    }
    public MyText(double x, double y, String text, Font font) {
        super(x, y, text);
        this.font = font;
        setFont(font);
    }
}
class SankeyDiagram extends Pane {
    private MyRectangle rectangle;
    private MyText labelTitle, diagramTitle;
    private Group recs, text, curves;

    public SankeyDiagram(String title, String label,
                                Map<String, Double> dataMap) {
        rectangle = new MyRectangle();
        labelTitle = new MyText();
        diagramTitle = new MyText();
        recs = new Group();
        text = new Group();
        curves = new Group();
        createSankeyDiagram(title, label, dataMap);
    }

    public void createSankeyDiagram(String title, String label,
                                            Map<String, Double> dataMap) {
        double paneWidth = getWidth();
        double paneHeight = getHeight();
        double Width = getWidth() * 0.2;
        double Height = getHeight() * 0.6;

        diagramTitle = new MyText(350, 550, title, Font.font("Courier",
FontWeight.BOLD, FontPosture.ITALIC, 25));

        rectangle = new MyRectangle(
                100, 100, 20, 300,
                Color.rgb(51, 166, 204, 0.5),
                Color.rgb(51, 166, 204, 0.5)
        );
        double totalHeight = rectangle.getHeight();
        double sum = calculateTotalSum(dataMap);
        label = label + " : " + sum;
        labelTitle = new MyText(
                rectangle.getX() + rectangle.getWidth() / 2,
                rectangle.getY() + rectangle.getHeight() / 2,
                label,
                Font.font("Courier", FontWeight.BOLD,
FontPosture.REGULAR, 20)
        );

        double recheheightchange = 0;
        double curveheightchange = 0;
        double X = rectangle.getX() + rectangle.getWidth();
        double Y = rectangle.getY();

        for (String key : dataMap.keySet()) {
            double currentheight = (dataMap.get(key) / sum) * totalHeight;

            int randomR = (int) (Math.random() * 256);
            int randomG = (int) (Math.random() * 256);
            int randomB = (int) (Math.random() * 256);

            MyRectangle r1 = new MyRectangle(
                    X + 600, Y - 60 + recheheightchange,
                    30, currentheight,
                    Color.rgb(randomR, randomG, randomB, 0.6),
                    Color.rgb(randomR, randomG, randomB, 0.6));

            recs.getChildren().add(r1);

            MyText type = new MyText(
                    r1.getX() + r1.getWidth() / 2,
                    r1.getY() + r1.getHeight() / 2,
                    key + " : " + dataMap.get(key),
                    Font.font("Courier", FontWeight.BOLD,
FontPosture.REGULAR, 18));
            text.getChildren().add(type);

            Path path = createPath(X, Y, curveheightchange, r1, currentheight);
            Color pathColor = Color.rgb(randomR, randomG, randomB, 0.4);
            path.setStroke(pathColor);
            path.setFill(pathColor);

            curves.getChildren().add(path);

            recheheightchange += currentheight + 30;
            curveheightchange += currentheight;
        }
        getChildren().addAll(diagramTitle, rectangle, recs, curves, labelTitle,
text);
    }

    public void changeColors() {
        List<MyRectangle> rectangles = new ArrayList<>();
        List<Path> paths = new ArrayList<>();

        for (Node node : recs.getChildren()) {
            if (node instanceof MyRectangle) {
                rectangles.add((MyRectangle) node);
            }
        }
        for (Node node : curves.getChildren()) {
            if (node instanceof Path) {
                paths.add((Path) node);
            }
        }
        for (int i = 0; i < rectangles.size(); i++) {
            MyRectangle rectangle = rectangles.get(i);
            Path path = paths.get(i);

            Color newColor = getRandomColor();
            rectangle.setStroke(newColor);
            rectangle.setFill(newColor);
            path.setStroke(newColor);
            path.setFill(newColor);
        }
    }

    private Color getRandomColor() {
        int randomR = (int) (Math.random() * 256);
        int randomG = (int) (Math.random() * 256);
        int randomB = (int) (Math.random() * 256);
        return Color.rgb(randomR, randomG, randomB, 0.4);
    }

    @Override
    public void setWidth(double width) {
        super.setWidth(width);
    }
    @Override
    public void setHeight(double heigth) {
        super.setHeight(heigth);
    }
    private double calculateTotalSum(Map<String, Double> dataMap) {
        double sum = 0;
        for (double value : dataMap.values()) {
            sum += value;
        }
        return sum;
    }
    private Path createPath(double x, double y,
                            double curveheightchange,
                            MyRectangle r1, double currentHeight) {
        //上曲线的起始点和控制点
        MoveTo moveTo1 = new MoveTo(x, y + curveheightchange);
        double endX = r1.getX();
        double endY = r1.getY();
        //Determine the amount of change at the control point 确定控制点的变化量
        double changeX = Math.abs(x - endX);
        double controlX1 = moveTo1.getX() + changeX / 3;
        double controlY1 = moveTo1.getY();
        double controlX2 = endX - changeX / 3;
        double controlY2 = endY;

        CubicCurveTo curveTo1 = new CubicCurveTo(
                controlX1, controlY1,
                controlX2, controlY2,
                endX, endY
        );
        //下曲线的起始点和控制点
        LineTo lineTo1 = new LineTo(r1.getX(), r1.getY() + r1.getHeight());
        double startX = x;
        double startY = y + curveheightchange + currentHeight;
        double controlX3 = startX + changeX / 3;
        double controlY3 = startY;
        double controlX4 = lineTo1.getX() - changeX / 3;
        double controlY4 = lineTo1.getY();

        CubicCurveTo curveTo2 = new CubicCurveTo(
                controlX4, controlY4,
                controlX3, controlY3,
                startX, startY
        );

        Path path = new Path();
        path.getElements().addAll(moveTo1, curveTo1, lineTo1, curveTo2);
        return path;
    }
}
```