**Самченко ІТІНФ-20-1 В-21**

Розробити програму, яка будуватиме на екрані графіки функцій sin(x), cos(x), x2 за вибором користувача.

**Class HelloApplication:**

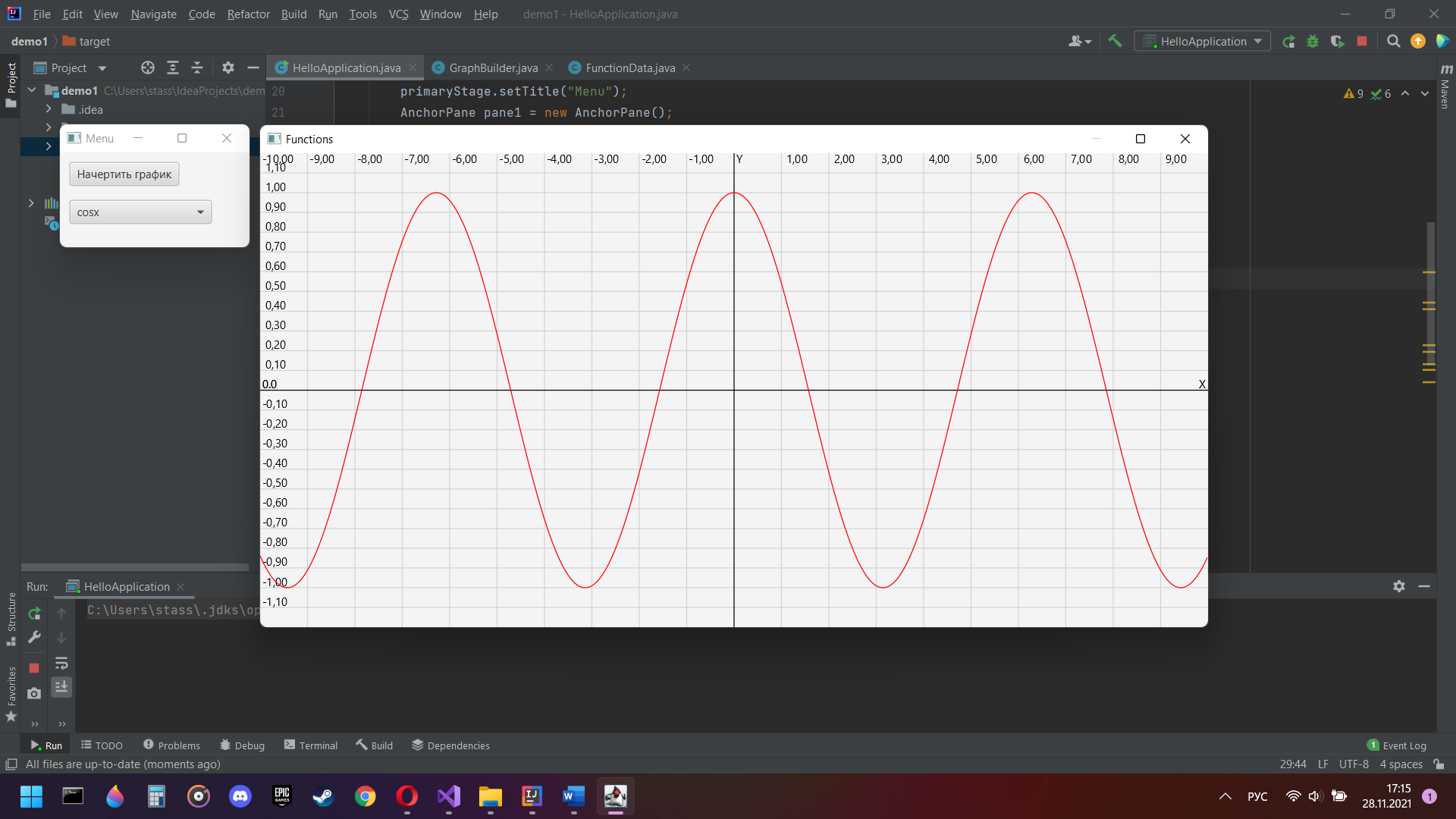
package com.example.demo1;  
  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.scene.control.Button;  
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.scene.control.ComboBox;  
import javafx.scene.layout.AnchorPane;  
import javafx.scene.paint.Color;  
import javafx.stage.Modality;  
import javafx.stage.Stage;  
import java.util.function.DoubleUnaryOperator;  
  
public class HelloApplication extends Application {  
 @Override  
 public void start(Stage primaryStage) {  
 primaryStage.setTitle("Menu");  
 AnchorPane pane1 = new AnchorPane();  
  
 Button button = new Button("Начертить график");  
 button.setLayoutX(10);  
 button.setLayoutY(10);  
  
 ObservableList<String> foo = FXCollections.*observableArrayList*("sinx", "cosx", "x^2");  
 ComboBox<String> comboBox = new ComboBox<String>(foo);  
 comboBox.setValue("Выберете функцию");  
 comboBox.setLayoutX(10);  
 comboBox.setLayoutY(50);  
  
 GraphBuilder builder = new GraphBuilder(pane1);  
 button.setOnAction(new EventHandler<ActionEvent>() {  
 @Override  
 public void handle(ActionEvent event) {  
 AnchorPane pane2 = new AnchorPane();  
 Scene secondScene = new Scene(pane2, 1000, 500);  
 GraphBuilder builder = new GraphBuilder(pane2);  
 if (comboBox.getValue() == "sinx") {  
 DoubleUnaryOperator sinx = x -> Math.*sin*(x);  
 builder.addFunction(sinx, Color.*BLUE*, 1);  
 } else if (comboBox.getValue() == "cosx") {  
 DoubleUnaryOperator cosx = x -> Math.*cos*(x);  
 builder.addFunction(cosx, Color.*RED*, 1);  
 } else if (comboBox.getValue() == "x^2") {  
 DoubleUnaryOperator x2 = x -> Math.*pow*(x, 2);  
 builder.addFunction(x2, Color.*GREEN*, 1);  
 }  
 builder.drawGraph(-10, 10);  
  
 Stage newWindow = new Stage();  
 newWindow.setTitle("Functions");  
 newWindow.setScene(secondScene);  
 newWindow.initModality(Modality.*WINDOW\_MODAL*);  
 newWindow.initOwner(primaryStage);  
 newWindow.setX(primaryStage.getX() + 200);  
 newWindow.setY(primaryStage.getY() + 100);  
 newWindow.show();  
 }  
 });  
  
 Scene scene = new Scene(pane1, 200, 100);  
 pane1.getChildren().addAll(button,comboBox);  
  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
  
 public static void main(String[] args) { *launch*(args); }  
}

**Class GraphBuilder:**

package com.example.demo1;  
  
import javafx.scene.layout.Pane;  
import javafx.scene.paint.Color;  
import javafx.scene.paint.Paint;  
import javafx.scene.shape.Line;  
import javafx.scene.shape.Polyline;  
import javafx.scene.text.Text;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.function.DoubleUnaryOperator;  
  
public class GraphBuilder {  
 // Number of steps to calculate the minimum and maximum of functions:  
 public static double *STEPS* = 100;  
 // Minimum indent from axes and graphics along the vertical axis:  
 public static double *VERTICAL\_GAP* = 0.2;  
 // The visibility threshold of the grid lines:  
 public static double *MIN\_PIXELS* = 20;  
 // Grid step threshold:  
 public static double *SCALE\_STEP* = 10;  
 // Minimum indent from axes:  
 public static double *MIN\_GAP* = 2;  
 // Indent from the edge of the picture:  
 public static double *MAX\_GAP* = 10;  
 // Number output format:  
 public static String *FORMAT* = "%5.2f";  
  
 // Approximate computation of the minimum of a function at a certain interval:  
 public static double minY(double from, double to, DoubleUnaryOperator operator) {  
 double min = operator.applyAsDouble(from);  
 double h = (to - from) / *STEPS*;  
 for (double x = from + h; x <= to; x += h) {  
 double y = operator.applyAsDouble(x);  
 if (min > y) {  
 min = y;  
 }  
 }  
 return min;  
 }  
  
 // Approximate calculation of the maximum of a function at a certain interval:  
 public static double maxY(double from, double to, DoubleUnaryOperator operator) {  
 double max = operator.applyAsDouble(from);  
 double h = (to - from) / *STEPS*;  
 for (double x = from + h; x <= to; x += h) {  
 double y = operator.applyAsDouble(x);  
 if (max < y) {  
 max = y;  
 }  
 }  
 return max;  
 }  
  
 // A panel that will display a graph:  
 private Pane pane;  
  
 // Colors of the grid and axes:  
 private Color gridColor = Color.*LIGHTGRAY*;  
 private Color axesColor = Color.*BLACK*;  
  
 // Functions to display:  
 private List<FunctionData> funcs = new ArrayList<>();  
  
 // In the constructor, we define the panel for plotting:  
 public GraphBuilder(Pane pane) {  
 this.pane = pane;  
 }  
  
 public Color getGridColor() {  
 return gridColor;  
 }  
  
 public void setGridColor(Color gridColor) {  
 this.gridColor = gridColor;  
 }  
  
 public Color getAxesColor() {  
 return axesColor;  
 }  
  
 public void setAxesColor(Color axesColor) {  
 this.axesColor = axesColor;  
 }  
  
 public void clearFunctions() {  
 funcs.clear();  
 }  
  
 public void addFunction(DoubleUnaryOperator operator, Paint paint, double width) {  
 FunctionData func = new FunctionData(operator, paint, width);  
 funcs.add(func);  
 }  
  
 // Plotting:  
 public void drawGraph(double xMin, double xMax) {  
 // Determine and calculate the range of the function:  
 double yMin = -*VERTICAL\_GAP*;  
 double yMax = *VERTICAL\_GAP*;  
 for (FunctionData func : funcs) {  
 double min = *minY*(xMin, xMax, func.getOperator());  
 double max = *maxY*(xMin, xMax, func.getOperator());  
 if (yMin > min) {  
 yMin = min;  
 }  
 if (yMax < max) {  
 yMax = max;  
 }  
 }  
 yMin -= *VERTICAL\_GAP* \* Math.*abs*(yMin);  
 yMax += *VERTICAL\_GAP* \* Math.*abs*(yMax);  
 // The size of the graph:  
 double width = pane.getWidth();  
 double height = pane.getHeight();  
 // Scales:  
 double xScale = width / (xMax - xMin);  
 double yScale = height / (yMax - yMin);  
 // Coordinates of the origin projection:  
 double x0 = -xMin \* xScale;  
 double y0 = yMax \* yScale;  
 pane.getChildren().clear();  
  
 // Grid:  
 double xStep = 1; // grid step  
 // Change the step if the grid lines are too frequent:  
 while (xStep \* xScale < *MIN\_PIXELS*)  
 xStep \*= *SCALE\_STEP*;  
 // Change the step if the grid lines are too rare:  
 while (xStep \* xScale > *MIN\_PIXELS* \* *SCALE\_STEP*)  
 xStep /= *SCALE\_STEP*;  
 // Vertical grid lines:  
 for (double dx = xStep; dx < xMax; dx += xStep) {  
 double x = x0 + dx \* xScale;  
 Line line = new Line(x, 0, x, height);  
 line.setStroke(gridColor);  
 pane.getChildren().add(line);  
 pane.getChildren().add(new Text(x + *MIN\_GAP*, *MAX\_GAP*, String.*format*(*FORMAT*, dx)));  
 }  
 for (double dx = -xStep; dx >= xMin; dx -= xStep) {  
 double x = x0 + dx \* xScale;  
 Line line = new Line(x, 0, x, height);  
 line.setStroke(gridColor);  
 pane.getChildren().add(line);  
 pane.getChildren().add(new Text(x + *MIN\_GAP*, *MAX\_GAP*, String.*format*(*FORMAT*, dx)));  
 }  
 double yStep = 1; // grid step  
 // Change the step if the grid lines are too frequent:  
 while (yStep \* yScale < *MIN\_PIXELS*) {  
 yStep \*= *SCALE\_STEP*;  
 }  
 // Change the step if the grid lines are too rare:  
 while (yStep \* yScale > *MIN\_PIXELS* \* *SCALE\_STEP*)  
 yStep /= *SCALE\_STEP*;  
 // Horizontal grid lines:  
 for (double dy = yStep; dy < yMax; dy += yStep) {  
 double y = y0 - dy \* yScale;  
 Line line = new Line(0, y, width, y);  
 line.setStroke(gridColor);  
 pane.getChildren().add(line);  
 pane.getChildren().add(new Text(*MIN\_GAP*, y - *MIN\_GAP*, String.*format*(*FORMAT*, dy)));  
 }  
 for (double dy = -yStep; dy > yMin; dy -= yStep) {  
 double y = y0 - dy \* yScale;  
 Line line = new Line(0, y, width, y);  
 line.setStroke(gridColor);  
 pane.getChildren().add(line);  
 pane.getChildren().add(new Text(*MIN\_GAP*, y - *MIN\_GAP*, String.*format*(*FORMAT*, dy)));  
 }  
  
 // Axes:  
 Line verticalAxis = new Line(x0, 0, x0, height);  
 verticalAxis.setStroke(axesColor);  
 pane.getChildren().add(verticalAxis);  
 Line horizontalAxis = new Line(0, y0, width, y0);  
 pane.getChildren().add(horizontalAxis);  
 pane.getChildren().add(new Text(*MIN\_GAP*, y0 - *MIN\_GAP*, "0.0"));  
 pane.getChildren().add(new Text(*MIN\_GAP*, y0 - *MIN\_GAP*, "0.0"));  
 pane.getChildren().add(new Text(width - *MAX\_GAP*, y0 - *MIN\_GAP*, "X"));  
 pane.getChildren().add(new Text(x0 + *MIN\_GAP*, *MAX\_GAP*, "Y"));  
  
 // Functions:  
 for (FunctionData func : funcs) {  
 Polyline polyline = new Polyline();  
 polyline.setStroke(func.getPaint());  
 polyline.setStrokeWidth(func.getWidth());  
 for (double x = 0; x < width; x++) {  
 double dx = (x - x0) / xScale;  
 double dy = func.getOperator().applyAsDouble(dx);  
 double y = y0 - dy \* yScale;  
 polyline.getPoints().addAll(x, y);  
 }  
 pane.getChildren().add(polyline);  
 }  
 }  
}

**Class FunctionData:**

package com.example.demo1;  
import javafx.scene.paint.Paint;  
import java.util.function.DoubleUnaryOperator;  
  
public class FunctionData {  
 private DoubleUnaryOperator operator;  
 private Paint paint;  
 private double width;  
  
 public FunctionData(DoubleUnaryOperator operator, Paint paint, double width) {  
 this.operator = operator;  
 this.paint = paint;  
 this.width = width;  
 }  
  
 public DoubleUnaryOperator getOperator() {  
 return operator;  
 }  
  
 public Paint getPaint() {  
 return paint;  
 }  
  
 public double getWidth() {  
 return width;  
 }  
}



Результат роботы