Додаток Б Тексти програмного коду

*студента групи ІП-91 І курсу*

*Дзюбака Д. П.*

(Обсяг програми (документа), арк., Кб)

*21 арк, 103 Кб*

(Вид носія даних)

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення вирішення задачі вирішення СЛАР*

*import* Model.Model;  
*import* Model.exception.InconsistentMatrixException;  
*import* Model.exception.InfiniteSolutionsAmountException;  
*import* Model.exception.InvalidInputException;  
*import* javafx.application.Application;  
*import* javafx.geometry.Insets;  
*import* javafx.scene.Scene;  
*import* javafx.scene.control.Button;  
*import* javafx.scene.control.ChoiceBox;  
*import* javafx.scene.control.Label;  
*import* javafx.scene.control.TextField;  
*import* javafx.scene.layout.\*;  
*import* javafx.stage.DirectoryChooser;  
*import* javafx.stage.FileChooser;  
*import* javafx.stage.Modality;  
*import* javafx.stage.Stage;  
  
*import* java.io.File;  
  
*public class* View *extends* Application {  
 Model model;  
 Stage st, alertStage, resultStage;  
 Scene scene, alertScene, resultScene;  
 BorderPane layout, alertLayout;  
 HBox topMenu, manChosePane, botMenu, genGenPane, genSlaeParamPane, readReadPane;  
 VBox readPane, manPane, genPane, alertPane, resultPane, genChoosePane;  
 AnchorPane buttonPane;  
 GridPane manFieldsPane;  
 Button solveB, genGenDoubleB, okB, genGenIntB, okRB, chooseInputFileB,chooseOutputDirB;  
 TextField inputFileF, manCoefsF[][], manFreeValsF[];  
 Label inputL, methodL, manVarNumL, readWarnL, genVarNumL, enterPathL, generateMatrixL, generatedMatrixL,  
 chooseFileFirstlyL, incorrectInputL, resultWrittenL, inputFieldEmptyL, fileDoesntExistL,  
 readMatrixL, inconsistentMatrixL, infiniteSolutionsL, invalidInputL, inputMatrixL,  
 genEqtNumL,manEqtNumL, inputFileL, outputFileL, inputFilePathL, outputFilePathL,  
 totalResultL;  
 ChoiceBox<String> inputTypeList, methodList;  
 ChoiceBox<Integer> manVarNumCB, genVarNumCB, genEqtNumCB, manEqtNumCB;  
 FileChooser fileChoose;  
 DirectoryChooser dirChoose;  
  
 *public static void* main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 *public void* start(Stage stage) *throws* Exception {  
 model = *new* Model();  
 st = stage;  
  
 st.setTitle("SLAE solver");  
 st.setWidth(580);  
 st.setHeight(500);  
  
 *//general menu* inputL = *new* Label("Choose input type : ");  
 methodL = *new* Label(" Choose solving method : ");  
 inputTypeList = *new* ChoiceBox<>();  
 inputTypeList.getItems().addAll("Generate randomly", "Read from file", "Enter manually");  
 inputTypeList.setValue("Enter manually");  
 inputTypeList.getSelectionModel().selectedItemProperty().addListener((v, oldVal, newVal) -> {  
 *if* (newVal == "Read from file") {  
 layout.setCenter(readPane);  
 scene.setRoot(layout);  
 } *else if* (newVal == "Generate randomly") {  
 layout.setCenter(genPane);  
 scene.setRoot(layout);  
 } *else if* (newVal == "Enter manually") {  
 layout.setCenter(manPane);  
 scene.setRoot(layout);  
 }  
 });  
 methodList = *new* ChoiceBox<>();  
 methodList.getItems().addAll("Gauss", "Jordan-Gauss", "Rotation");  
 methodList.setValue("Gauss");  
 outputFileL = *new* Label("Output file :");  
 outputFilePathL = *new* Label("unselected");  
 topMenu = *new* HBox(10);  
 topMenu.getChildren().addAll(inputL, inputTypeList, methodL, methodList);  
 topMenu.setPadding(*new* Insets(10, 10, 10, 10));  
 totalResultL = *new* Label();  
  
 chooseOutputDirB = *new* Button("Choose directory");  
 chooseOutputDirB.setOnAction(ae ->{  
 dirChoose = *new* DirectoryChooser();  
 File outputDir = dirChoose.showDialog(*null*);  
  
 *if*(outputDir != *null*)  
 outputFilePathL.setText(outputDir.getAbsolutePath());  
 });  
  
 solveB = *new* Button("Solve");  
 solveB.setOnAction(actionEvent -> {  
 *try* {  
 model.nullifyResultingStrings();  
 *if* (inputTypeList.getValue() == "Generate randomly" && !model.matrixIsReady()) {  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, generateMatrixL);  
 alertStage.show();  
 } *else if* (inputTypeList.getValue() == "Enter manually")   
 makeMatrixManually();  
 model.solveMatrixByMethod(methodList.getValue());  
 *if* (model.getVarNum() == 2 && model.getEquationsNum() == 2)  
 getChartSolving(model.getOriginSlae());  
 *if* (model.resultExists()) {  
 totalResultL.setText(model.makeTotalResultString());  
 setResultStage();  
 *if* (!outputFilePathL.getText().equals("unselected"))  
 model.writeResultToFile(outputFilePathL.getText());  
  
 resultWrittenL.setText("Result was written to file : " + outputFilePathL.getText());  
 resultPane.getChildren().remove(1);  
 resultPane.getChildren().add(1, totalResultL);  
 resultStage.show();  
 }  
  
 } *catch* (InconsistentMatrixException e) {  
 inconsistentMatrixL.setText(e.getMessage());  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, inconsistentMatrixL);  
 alertStage.setMinWidth(inconsistentMatrixL.getLayoutBounds().getWidth());  
 alertStage.show();  
 } *catch* (InfiniteSolutionsAmountException e) {  
 infiniteSolutionsL.setText(e.getMessage());  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, infiniteSolutionsL);  
 alertStage.setMinWidth(infiniteSolutionsL.getLayoutBounds().getWidth());  
 alertStage.show();  
 } *catch* (InvalidInputException e) {  
 invalidInputL.setText(e.getMessage());  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, invalidInputL);  
 alertStage.setMinWidth(invalidInputL.getLayoutBounds().getWidth());  
 alertStage.show();  
 }  
 });  
 botMenu = *new* HBox(10);  
 botMenu.getChildren().addAll(outputFileL, outputFilePathL, chooseOutputDirB, solveB);  
 buttonPane = *new* AnchorPane();  
 buttonPane.getChildren().addAll(botMenu);  
 AnchorPane.*setRightAnchor*(botMenu, 10d);  
 AnchorPane.*setBottomAnchor*(botMenu, 10d);  
 layout = *new* BorderPane();  
 layout.setTop(topMenu);  
 layout.setBottom(buttonPane);  
 scene = *new* Scene(layout);  
  
 *//result box;  
  
 // alertBox* alertStage = *new* Stage();  
 alertStage.initModality(Modality.*APPLICATION\_MODAL*);  
 alertStage.setTitle("Alert box");  
 alertStage.setHeight( 160);  
 alertStage.setWidth(240 );  
 enterPathL = *new* Label("Enter valid input file path!");  
 enterPathL.setPadding(*new* Insets(10, 10, 10, 10));  
 chooseFileFirstlyL = *new* Label("Enter valid output file path!");  
 incorrectInputL = *new* Label("Input matrix file must contain only " +  
 "\ndecimal or float(using '.') values " +  
 "\nfollowed by '-' if needed");  
 inputFieldEmptyL = *new* Label("Input file field is empty");  
 fileDoesntExistL = *new* Label("Specified file doesn't exist");  
 generateMatrixL = *new* Label("Generate matrix firstly!");  
 inconsistentMatrixL = *new* Label("Matrix is inconsistent");  
 infiniteSolutionsL = *new* Label("Matrix has infinite solutions amount");  
 invalidInputL = *new* Label("Input matrix characters must contain only " +  
 "\ndecimal or floating point(using '.') numbers " +  
 "\nfollowed by '-' if need");  
 okB = *new* Button("OK");  
 okB.setOnAction(ae -> {  
 alertStage.close();  
 });  
 alertPane = *new* VBox();  
 alertPane.getChildren().add(0, enterPathL);  
 alertPane.getChildren().add(1, okB);  
 VBox.*setMargin*(okB, *new* Insets(10, 50, 10, 10));  
 alertLayout = *new* BorderPane();  
 alertLayout.setCenter(alertPane);  
 alertScene = *new* Scene(alertLayout);  
 alertStage.setScene(alertScene);  
  
 *// "generate" pane* genVarNumL = *new* Label("Set variables amount : ");  
 genVarNumCB = *new* ChoiceBox<>();  
 genVarNumCB.getItems().addAll(2, 3, 4, 5, 6, 7, 8);  
 genVarNumCB.setValue(2);  
 genEqtNumL = *new* Label("Set equations amount : ");  
 genEqtNumCB = *new* ChoiceBox<>();  
 genEqtNumCB.getItems().addAll(2, 3, 4, 5, 6, 7, 8);  
 genEqtNumCB.setValue(2);  
 genGenDoubleB = *new* Button("Generate float num");  
 genGenDoubleB.setOnAction(ae -> {  
 model.generateFloatMatrix(genVarNumCB.getValue(), genEqtNumCB.getValue());  
 genPane.getChildren().remove(generatedMatrixL);  
 generatedMatrixL = *getLabeledMatrix*(model.getOriginSlae());  
 genPane.getChildren().add(generatedMatrixL);  
 layout.setCenter(genPane);  
 });  
 genGenIntB = *new* Button("Generate integer num");  
 genGenIntB.setOnAction(ae -> {  
 model.generateIntMatrix(genVarNumCB.getValue(), genEqtNumCB.getValue());  
 genPane.getChildren().remove(generatedMatrixL);  
 generatedMatrixL = *getLabeledMatrix*(model.getOriginSlae());  
 genPane.getChildren().add(generatedMatrixL);  
 layout.setCenter(genPane);  
 });  
 generatedMatrixL = *new* Label();  
 VBox.*setMargin*(generatedMatrixL, *new* Insets(10,10,10,10));  
 genGenPane = *new* HBox();  
 genSlaeParamPane = *new* HBox();  
 genChoosePane = *new* VBox();  
 genChoosePane.setPadding(*new* Insets(10));  
 genSlaeParamPane.getChildren().addAll(genVarNumL, genVarNumCB, genEqtNumL, genEqtNumCB);  
 genGenPane.getChildren().addAll(genGenDoubleB, genGenIntB);  
 genChoosePane.getChildren().addAll(genSlaeParamPane, genGenPane);  
 genPane = *new* VBox();  
 genPane.getChildren().addAll(genChoosePane, generatedMatrixL);  
  
 *// "readPane"* readWarnL = *new* Label("WARNINGS :\n - coefficients in file have to be separated by spaces\n" +  
 " - each coefficient including '0' have to be specified");  
 fileChoose = *new* FileChooser();  
 chooseInputFileB = *new* Button("Choose file");  
 chooseInputFileB.setOnAction(ae -> {  
 File inputF = fileChoose.showOpenDialog(*null*);  
  
 *if*(inputF != *null*)  
 *if*(model.fileIsValid(inputF)){  
 readPane.getChildren().removeAll(readMatrixL);  
 readMatrixL = *getLabeledMatrix*(model.getOriginSlae());  
 readPane.getChildren().add(readMatrixL);  
 readReadPane.getChildren().remove(1);  
 inputFilePathL.setText(inputF.getAbsolutePath());  
 readReadPane.getChildren().add(1, inputFilePathL);  
 solveB.setDisable(*false*);  
  
 } *else* {  
 *if*(!solveB.isDisabled())  
 solveB.setDisable(*true*);  
 alertPane.getChildren().remove(0);  
 alertPane.getChildren().add(0, incorrectInputL);  
 alertStage.show();  
 }  
 });  
 readWarnL.setPadding(*new* Insets(10));  
 inputFileF = *new* TextField();  
 inputFileF.setPromptText("Enter input SLAE matrix file path : ");  
 inputFileL = *new* Label("Input file : ");  
 inputFilePathL = *new* Label("unselected");  
 readMatrixL = *new* Label("");  
 readReadPane = *new* HBox();  
 readReadPane.setPadding(*new* Insets(3, 3, 3, 3));  
 readReadPane.getChildren().addAll(inputFileL,inputFilePathL, chooseInputFileB);  
 HBox.*setMargin*(inputFileL, *new* Insets(5,5,5,5));  
 HBox.*setMargin*(inputFilePathL, *new* Insets(5,5,5,5));  
 HBox.*setMargin*(chooseInputFileB, *new* Insets(5,5,5,5));  
 readPane = *new* VBox();  
 readPane.setPadding(*new* Insets(3, 3, 3, 3));  
 readPane.getChildren().addAll(readWarnL, readReadPane,readMatrixL);  
  
 *//"manual" choose pane* manVarNumL = *new* Label("Choose variables amount : ");  
 manVarNumCB = *new* ChoiceBox<>();  
 manVarNumCB.getItems().addAll(2, 3, 4, 5, 6, 7, 8);  
 manVarNumCB.setValue(2);  
 manEqtNumL = *new* Label("Choose equations amount : ");  
 manEqtNumCB = *new* ChoiceBox<>();  
 manEqtNumCB.getItems().addAll(2, 3, 4, 5, 6, 7, 8);  
 manEqtNumCB.setValue(2);  
 manVarNumCB.getSelectionModel().selectedItemProperty().addListener((v, oldval, newVal) -> {  
 manPane.getChildren().remove(manFieldsPane);  
 manFieldsPane = *new* GridPane();  
 manFieldsPane.setPadding(*new* Insets(10));  
 manFieldsPane.setHgap(5);  
 manFieldsPane.setVgap(10);  
 setManualGridPane(newVal, manEqtNumCB.getValue());  
 manPane.getChildren().add(manFieldsPane);  
 layout.setCenter(manPane);  
 });  
 manEqtNumCB.getSelectionModel().selectedItemProperty().addListener((v, oldval, newVal) -> {  
 manPane.getChildren().remove(manFieldsPane);  
 manFieldsPane = *new* GridPane();  
 manFieldsPane.setPadding(*new* Insets(10));  
 manFieldsPane.setHgap(5);  
 manFieldsPane.setVgap(10);  
 setManualGridPane(manVarNumCB.getValue(),newVal);  
 manPane.getChildren().add(manFieldsPane);  
 layout.setCenter(manPane);  
 });  
 manChosePane = *new* HBox(10);  
 manChosePane.setPadding(*new* Insets(10, 10, 10, 10));  
 manChosePane.getChildren().addAll(manVarNumL, manVarNumCB, manEqtNumL, manEqtNumCB);  
 *//"manual" elements pane* manFieldsPane = *new* GridPane();  
 manFieldsPane.setPadding(*new* Insets(10));  
 manFieldsPane.setHgap(5);  
 manFieldsPane.setVgap(10);  
 setManualGridPane(manVarNumCB.getValue(), genEqtNumCB.getValue());  
 manPane = *new* VBox();  
 manPane.setPadding(*new* Insets(10));  
 manPane.getChildren().addAll(manChosePane, manFieldsPane);  
 layout.setCenter(manPane);  
  
 st.setScene(scene);  
 st.show();  
 }  
 *public void* getChartSolving(*double*[][] matrix) {  
 *if* (matrix.length != 2 & matrix[0].length != 3)  
 *throw new* RuntimeException("Matrix has less or more than two variables to solve it by chart");  
 *else* {  
 ChartSolve cs = *new* ChartSolve(matrix);  
 cs.start(*new* Stage());  
 }  
 }  
 *public void* makeMatrixManually() {  
 *int* length = manCoefsF.length;  
 *int* width = manCoefsF[0].length+1;  
 *double*[][] matrix = *new double*[length][width];  
 String var;  
 *for* (*int* i = 0; i < length; i++)  
 *for* (*int* j = 0; j < width; j++) {  
 *if* (j < width-1 && gridInputIsValid(var = manCoefsF[i][j].getText().trim()))  
 matrix[i][j] = Double.*parseDouble*(var);  
 *else if*(j == width-1 && gridInputIsValid(var = manFreeValsF[i].getText().trim()))  
 matrix[i][j] = Double.*parseDouble*(var);  
 *else  
 throw new* InvalidInputException("Matrix grid must contain only decimal or float value followed by '-' if needed");  
 }  
 model.setMatrix(matrix);  
 }  
 *private boolean* gridInputIsValid(String s) {  
 *return* s.matches("-?\\d+([.,]\\d+)?");  
 }  
 *public void* setManualGridPane(*int* varNum, *int* eqtNum) {  
 manCoefsF = *new* TextField[eqtNum][varNum];  
 manFreeValsF = *new* TextField[eqtNum];  
 *for* (*int* i = 0; i < eqtNum; i++) {  
 *for* (*int* j = 0; j < varNum; j++) {  
 manCoefsF[i][j] = *new* TextField("0");  
 manCoefsF[i][j].setMaxSize(30, 30);  
 manFieldsPane.getChildren().add(manCoefsF[i][j]);  
 GridPane.*setConstraints*(manCoefsF[i][j], j, i);  
 *if* (j + 2 == varNum+1) {  
 manFreeValsF[i] = *new* TextField("0");  
 manFreeValsF[i].setMaxSize(30, 30);  
 manFieldsPane.getChildren().add(manFreeValsF[i]);  
 GridPane.*setConstraints*(manFreeValsF[i], j + 1, i);  
 GridPane.*setMargin*(manFreeValsF[i], *new* Insets(0, 10, 0, 10));  
 }  
 }  
 }  
 }  
  
 *public static* Label getLabeledMatrix(*double*[][] matrix) {  
 Label label = *new* Label("");  
 *if* (matrix[0].length == 3) {  
 *for* (*int* i = 0; i < matrix.length; i++)  
 label.setText(label.getText() + matrix[i][0] + " x" + (i+1) + " " + (matrix[i][1] >= 0 ? "+" + matrix[i][1] : matrix[i][1])  
 + " y" + (i+1) + " = " + matrix[i][2] + "\n");  
  
 }*else  
 for* (*int* i = 0; i < matrix.length; i++) {  
 *for* (*int* j = 0; j < matrix[0].length; j++)  
 *if* (j < matrix[0].length - 1) {  
 *int* index = j + 1;  
 *if* (j < matrix[0].length - 2)  
 label.setText(label.getText() + String.*format*(" %.2f x" + index + " + ", matrix[i][j]));  
 *else* label.setText(label.getText() + String.*format*(" %.2f x" + index, matrix[i][j]));  
 } *else* label.setText(label.getText() + String.*format*(" = %.2f " + "%n", matrix[i][j]));  
 }  
 *return* label;  
 }  
 *private void* setResultStage() {  
 resultStage = *new* Stage();  
 resultStage.setTitle("Result");  
 resultStage.setHeight(500);  
 resultStage.setWidth(500);  
 resultWrittenL = *new* Label();  
 okRB = *new* Button("OK");  
 okRB.setOnAction(ae -> {  
 resultStage.close();  
 });  
 resultPane = *new* VBox();  
 resultPane.getChildren().addAll(resultWrittenL, totalResultL, okRB);  
 VBox.*setMargin*(okRB, *new* Insets(10, 50, 10, 50));  
 resultScene = *new* Scene(resultPane);  
 resultStage.setScene(resultScene);  
 }  
}

*package* Model;  
  
*import* Model.exception.InconsistentMatrixException;  
*import* Model.exception.InfiniteSolutionsAmountException;  
  
*import* java.io.\*;  
*import* java.math.BigDecimal;  
*import* java.util.LinkedList;  
*import* java.util.*List*;  
  
*public class* Model {  
 *private* BufferedReader br;  
 *private* BufferedWriter bw;  
 *private double*[][] originSlae;  
 *private int* eqtNum;  
 *private int* varNum;  
 *private int* consistence;  
 *private double*[][] resultSlae;  
 *private double*[] result;  
 *private* String methodName;  
 *private* String originSlaeString, resultSlaeString, resultString, arithmeticsString, totalResultString;  
  
 *private static int sumCount* = 0, *subtrCount* = 0, *multCount* = 0, *divCount* = 0;  
 *public boolean* fileIsValid(File input) {  
 *List*<String[]> tempList = *new* LinkedList<>();  
 *try* {  
 br = *new* BufferedReader(*new* FileReader(input));  
 *while* (br.ready())  
 tempList.add(br.readLine().split(" +"));  
 br.close();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
 *int* maxWidth = 0;  
 *for* (*int* i = 0; i < tempList.size(); i++)  
 *if*(tempList.get(i)[0].isEmpty() | tempList.get(i)[0].isBlank())  
 tempList.remove(i);  
 *else* {  
 *for* (String s:tempList.get(i)) {  
 *if*(!s.matches("-?\\d+([.,]\\d+)?"))  
 *return false*;  
 *else if* (s.contains(","))  
 s.replace(',', '.');  
 }  
 *if* (tempList.get(i).length > maxWidth)  
 maxWidth = tempList.get(i).length;  
 }  
 *int* eqtNum = 2, eqtWidth = 2;  
 *if* (tempList.size() > eqtNum)  
 eqtNum = tempList.size();  
 *if* (maxWidth > eqtWidth)  
 eqtWidth = maxWidth;  
 originSlae = *new double*[eqtNum][eqtWidth];  
 *for* (*int* i = 0; i < eqtNum; i++) {  
 originSlae[i] = getEquationCoefs(tempList.get(i), eqtWidth);  
 }  
 *return true*;  
 }  
 *private double*[] getEquationCoefs(String[] s, *int* eqtWidth) {  
 *double*[] coefs = *new double*[eqtWidth];  
 *for* (*int* i = 0; i < eqtWidth; i++)  
 *if* (i < s.length && !s[i].isEmpty() & !s[i].isBlank())  
 coefs[i] = Double.*parseDouble*(s[i]);  
 *else* coefs[i] = 0;  
 *return* coefs;  
 }  
 *public void* writeResultToFile(String outPath) {  
 File outF = *new* File(outPath);  
 *if* (outF.isDirectory()) {  
 outF = *new* File(outPath + "\\" + (outF.listFiles().length+1) + ".txt");  
 *try* {  
 outF.createNewFile();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 *try* {  
 bw = *new* BufferedWriter(*new* FileWriter(outF));  
 bw.write(totalResultString);  
 bw.flush();  
 bw.close();  
 } *catch* (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 *public* String makeTotalResultString() {  
 *return* totalResultString =  
 "Method: "+ methodName + "\n" +  
 "Input slae :\n" +  
 makeOriginSlaeString() +  
 "\nResult slae : \n" +  
 makeResultSlaeString() +  
 "\nResult :\n" +  
 makeResultString()+  
 makeArithmeticsString();  
 }  
 *private* String makeArithmeticsString() {  
 *return* arithmeticsString = "Additions: " + *sumCount* + "\n" +  
 "Subtractions: " + *subtrCount* + "\n" +  
 "Multiplies: " + *multCount* + "\n" +  
 "Divisions: " + *divCount* + "\n";  
 }  
 *private* String makeOriginSlaeString() {  
 *if* (originSlae[0].length == 3) {  
 *for* (*int* i = 0; i < originSlae.length; i++) {  
 originSlaeString += originSlae[i][0] + " x" + (i + 1) + " " + (originSlae[i][1] >= 0 ? "+" + originSlae[i][1] : originSlae[i][1])  
 + " y" + (i + 1) + " = " + originSlae[i][2] + "\n";  
 }  
 } *else* {  
 *for* (*int* i = 0; i < originSlae.length; i++)  
 *for* (*int* j = 0; j < originSlae[0].length; j++) {  
 *int* index = j + 1;  
 *if* (j < originSlae[0].length - 1) {  
 *if* (j < originSlae[0].length - 2) {  
 originSlaeString += String.*format*("%.2f x" + index + (originSlae[i][j + 1] >= 0 ? " +" : " "), originSlae[i][j]);  
 } *else* {  
 originSlaeString += String.*format*(" %.2f x" + index + " ", originSlae[i][j]);  
 }  
 } *else* {  
 originSlaeString += String.*format*(" = %.2f " + "%n", originSlae[i][j]);  
 }  
 }  
 }  
 *return* originSlaeString;  
 }  
 *private* String makeResultSlaeString() {  
 *for* (*int* i = 0; i < resultSlae.length; i++)  
 *for* (*int* j = 0; j < resultSlae[0].length; j++) {  
 *int* index = j + 1;  
 *if* (j < resultSlae[0].length - 1)  
 *if* (j < resultSlae[0].length - 2)  
 resultSlaeString += String.*format*("%.2f x" + index + (originSlae[i][j + 1] >= 0 ? " +" : " "), resultSlae[i][j]);  
 *else* resultSlaeString += String.*format*(" %.2f x" + index, resultSlae[i][j]);  
 *else* resultSlaeString += String.*format*(" = %.2f " + "%n", resultSlae[i][j]);  
 }  
 *return* resultSlaeString;  
 }  
 *private* String makeResultString() {  
 *if* (result.length == 2)  
 resultString += String.*format*("x" + " = " + " %.3f \n", result[0]) + String.*format*("y" + " = " + " %.3f \n", result[1]);  
 *else  
 for* (*int* i = 0; i < result.length; i++) {  
 *int* index = i + 1;  
 resultString += String.*format*("x" + index + " = " + " %.3f \n", result[i]);  
 }  
 *return* resultString;  
  
 }  
 *public void* generateFloatMatrix(*int* varNum, *int* eqtNum) {  
 *double*[][] matrix = *new double*[eqtNum][varNum + 1];  
 *for* (*int* i = 0; i < eqtNum; i++)  
 *for* (*int* j = 0; j < varNum + 1; j++)  
 matrix[i][j] = Math.*random*() \* 10;  
 *this*.originSlae = matrix;  
 *this*.varNum = varNum;  
 *this*.eqtNum = eqtNum;  
 }  
 *public void* generateIntMatrix(*int* varNum, *int* eqtNum) {  
 *double*[][] matrix = *new double*[eqtNum][varNum + 1];  
 *for* (*int* i = 0; i < eqtNum; i++)  
 *for* (*int* j = 0; j < varNum + 1; j++)  
 matrix[i][j] = Math.*round*(Math.*random*() \* 10);  
 *this*.originSlae = matrix;  
 *this*.varNum = varNum;  
 *this*.eqtNum = eqtNum;  
 }  
 *public static double*[][] toDoubleMatrix(BigDecimal[][] m) {  
 *double*[][] res = *new double*[m.length][m[0].length];  
 *for* (*int* i = 0; i < m.length; i++)  
 *for* (*int* j = 0; j < m[0].length; j++)  
 res[i][j] = m[i][j].doubleValue();  
  
 *return* res;  
 }  
 *public static* BigDecimal[][] toBigDecimalMatrix(*double*[][] m) {  
 BigDecimal[][] res = *new* BigDecimal[m.length][m[0].length];  
 *for* (*int* i = 0; i < m.length; i++)  
 *for* (*int* j = 0; j < m[0].length; j++)  
 res[i][j] = BigDecimal.*valueOf*(m[i][j]);  
  
 *return* res;  
 }  
 *private void* nullifyCounters() {  
 *sumCount* = 0;  
 *subtrCount* = 0;  
 *multCount* = 0;  
 *divCount* = 0;  
 }  
 *public void* solveMatrixByMethod(String methodName) {  
 *if* (matrixIsReady()) {  
 *this*.eqtNum = originSlae.length;  
 *this*.varNum = originSlae[0].length - 1;  
 *if* ((*this*.consistence = *getMatrixConsistence*(*cloneMatrix*(originSlae))) == -1) {  
 *throw new* InconsistentMatrixException("Matrix is inconsistent");  
 } *else if* (*this*.consistence == 1)  
 *throw new* InfiniteSolutionsAmountException("Matrix has infinite solutions amount");  
 }  
 *this*.methodName = methodName;  
 nullifyCounters();  
 *if* (methodName == "Gauss") {  
 result = gaussSolve(*cloneMatrix*(originSlae));  
 }  
 *else if* (methodName == "Jordan-Gauss") {  
 result = jordanGaussSolve(*cloneMatrix*(originSlae));  
 }  
 *else if* (methodName == "Rotation") {  
 result = rotationSolve(*cloneMatrix*(originSlae));  
 }  
 }  
 *public double*[] gaussSolve(*double*[][] matrix) {  
 *makeTriangleView*(matrix);  
 *return* backtrace(matrix);  
 }  
 *public static void* makeTriangleView(*double*[][] matrix) {  
 *int* eqtNum = matrix.length;  
 *int* eqtWidth = matrix[0].length - 1;  
  
 *for* (*int* i = 0; i < eqtNum; i++) {  
 *int* max = i;  
 *for* (*int* j = i+1; j < eqtNum; j++)  
 *if* (Math.*abs*(matrix[j][i]) < Math.*abs*(matrix[max][j]))  
 max = j; *double*[] temp = matrix[i];  
 matrix[i] = matrix[max];  
 matrix[max] = temp; *for* (*int* k = i+1; k < eqtNum; k++) {  
 *if* (matrix[i][i] == 0) {  
 *addOneToEachElement*(matrix[i]);  
 *sumCount* += matrix[i].length;  
 }  
 *double* alfa = matrix[k][i] / matrix[i][i];  
 *divCount*++;  
matrix[k][eqtWidth] -= alfa \* matrix[i][eqtWidth];  
 *multCount*++;  
 *subtrCount*++;  
  
 *for* (*int* j = i; j < eqtNum; j++) {  
 matrix[k][j] -= alfa\*matrix[i][j];  
 *multCount*++;  
 *subtrCount*++;  
 }  
 }  
 }  
 }  
 *private static void* addOneToEachElement(*double*[] arr) {  
 *for* (*int* i = 0; i < arr.length; i++) {  
 arr[i] += 1;  
 }  
 }  
 *private double*[] backtrace(*double*[][] matrix) {  
 *int* length = matrix.length;  
 *int* width = matrix[0].length;  
 *double*[] x = *new double*[length];  
 *for* (*int* i = length-1; i >= 0; i--) {  
 *double* sum = 0;  
 *for* (*int* j = i+1; j < length; j++) {  
 sum += matrix[i][j] \* x[j];  
 *multCount*++;  
 *sumCount*++;  
 }x[i] = (matrix[i][width-1] - sum)/matrix[i][i];  
 *subtrCount*++;  
 *divCount*++;  
 }  
 *this*.resultSlae = matrix;  
 *return* x;  
 }  
 *public double*[] jordanGaussSolve(*double*[][] matrix) {  
 *int* n = matrix.length;  
 *if* ((matrix[0].length - matrix.length) != 1)  
 *throw new* RuntimeException("Coefficient matrix is not square!");  
 *if* (matrix[0][0] == 0) {  
 *for* (*int* i = 0; i < varNum; i++) {  
 matrix[0][i] += 1;  
 }  
 }  
 *for* (*int* i = 0; i < n; i++) {  
 *double* permittingElement = matrix[i][i];  
  
 *for* (*int* j = 0; j < n+1; j++) { *//making permitting element equal to 1* matrix[i][j] /= permittingElement;  
 *divCount*++;  
 }  
 permittingElement = matrix[i][i];  
 *for* (*int* j = i+1; j < matrix[0].length; j++)  
 *for* (*int* k = 0; k < n; k++)  
 *if* (k != i) {  
 matrix[k][j] = permittingElement \* matrix[k][j] - matrix[i][j] \* matrix[k][i];  
 *multCount* += 2;  
 *subtrCount*++;  
 }  
  
 *for* (*int* j = 0; j < n; j++)  
 *if* (i != j)  
 matrix[j][i] = 0;  
 }  
  
 *double*[] x = *new double*[n];  
 *for* (*int* j = 0; j < n; j++)  
 x[j] = matrix[j][n];  
 *this*.resultSlae = matrix;  
  
 *return* x;  
 }  
  
 *public double*[] rotationSolve(*double*[][] matrix) {  
 *double* c,s, mik;  
 *for* (*int* i = 0 ; i < matrix.length; i++) {  
 *for* (*int* j = i+1; j < matrix.length; j++) {  
 c = matrix[i][i];  
 s = matrix[j][i];  
 *for* (*int* k = 0; k < matrix[0].length; k++) {  
 mik = matrix[i][k];  
 matrix[i][k] = c \* matrix[i][k] + s \* matrix[j][k];  
 matrix[j][k] = c \* matrix[j][k] - s \* mik;  
 *multCount* += 4;  
 *sumCount*++;  
 *subtrCount*++;  
 }  
 }  
 }  
 *double* sum;  
 *double*[] x = *new double*[varNum];  
 *for* (*int* i = varNum-1; i >= 0; i--) {  
 sum = 0;  
 *for* (*int* j = i+1; j < eqtNum; j++) {  
 sum += matrix[i][j] \* x[j];  
 *multCount*++;  
 *sumCount*++;  
 }  
 x[i] = (matrix[i][varNum] - sum)/matrix[i][i];  
  
 *subtrCount*++;  
 *divCount*++;  
 }  
 *this*.resultSlae = matrix;  
  
 *return* x;  
 }  
 *public static int* getMatrixConsistence(*double*[][] m) {  
 *makeTriangleView*(m);  
 *int* extendedMatrixRank = *getExtendedMatrixRank*(m);  
 *if*(extendedMatrixRank < m[0].length-1)  
 *return* 1;  
 *return* Integer.*compare*(*getRegularMatrixRank*(m), extendedMatrixRank);  
 }  
 *private static int* getExtendedMatrixRank(*double*[][] m) {  
 *int* nonZeroRawCounter = 0;  
 *for* (*int* i = 0; i < m.length; i++) {  
 *for* (*int* j = 0; j < m[0].length; j++) {  
 *if*(m[i][j] != 0){  
 nonZeroRawCounter++;  
 j = m[0].length;  
 }  
 }  
 }  
 *return* nonZeroRawCounter;  
 }  
 *private static int* getRegularMatrixRank(*double*[][] m) {  
 *int* nonZeroRawCounter = 0;  
 *for* (*int* i = 0; i < m.length; i++) {  
 *for* (*int* j = 0; j < m[0].length-1; j++) {  
 *if*(m[i][j] != 0){  
 nonZeroRawCounter++;  
 j = m[0].length;  
 }  
 }  
 }  
 *return* nonZeroRawCounter;  
 }  
 *public void* nullifyResultingStrings() {  
 originSlaeString = "";  
 resultSlaeString = "";  
 resultString = "";  
 arithmeticsString = "";  
 totalResultString = "";  
 }  
 *public boolean* matrixIsReady() {  
 *return this*.originSlae != *null*;  
 }  
 *public static double*[][] cloneMatrix(*double*[][] matrix) {  
 *double*[][] result = *new double*[matrix.length][matrix[0].length];  
 *for* (*int* i = 0; i < matrix.length; i++)  
 *for* (*int* j = 0; j < matrix[0].length; j++)  
 result[i][j] = matrix[i][j];  
 *return* result;  
 }  
 *public int* getVarNum() {  
 *return* varNum;  
 }  
 *public int* getEquationsNum() {  
 *return* eqtNum;  
 }  
 *public void* setMatrix(*double*[][] matrix) {  
 *this*.originSlae = matrix;  
 *this*.varNum = matrix[0].length - 1;  
 *this*.eqtNum = matrix.length;  
 }  
 *public boolean* resultExists() {  
 *return* result != *null*;  
 }  
 *public double*[][] getOriginSlae() {  
 *return* originSlae;  
 }  
}

*import* Model.Model;  
*import* javafx.application.Application;  
*import* javafx.scene.Node;  
*import* javafx.scene.Scene;  
*import* javafx.scene.layout.AnchorPane;  
*import* javafx.scene.layout.Pane;  
*import* javafx.scene.layout.StackPane;  
*import* javafx.scene.paint.Color;  
*import* javafx.scene.shape.Line;  
*import* javafx.scene.shape.Rectangle;  
*import* javafx.scene.shape.Shape;  
*import* javafx.scene.text.Font;  
*import* javafx.scene.text.Text;  
*import* javafx.stage.Stage;  
  
*import* java.math.BigDecimal;  
*import* java.math.RoundingMode;  
  
*public class* ChartSolve *extends* Application {  
 *private* Stage stage;  
 *private* Scene scene;  
 *private* StackPane root;  
 *private* Pane worldPane;  
 *private* String title = "Chart Solve";  
 *private double* height =720;  
 *private double* width = 720;  
  
 *private double*[][] slae;  
 *private* Coordinates screenResult, worldResult;  
  
 *private* Line l1;  
 *private* Line l2;  
 *private* Shape intersection;  
  
 *private double* worldMinX = -10;  
 *private double* worldMaxX = 10;  
 *private double* worldMinY = -10;  
 *private double* worldMaxY = 10;  
  
 *private double* offsetX = 0;  
 *private double* offsetY = 0;  
  
 *private double* scaleX = 1.0;  
 *private double* scaleY = 1.0;  
  
 *public static void* main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 *public void* start(Stage stage) {  
 *this*.stage = stage;  
 prepareWindow();  
 scaleX = 10;  
 scaleY = 10;  
  
 offsetX = -width/2/scaleX;  
 offsetY = -height/2/scaleY;  
 update();  
 }  
  
 *public* ChartSolve(*double*[][] matrix) {  
 *this*.slae = matrix;  
 }  
  
 *public* ChartSolve() {  
  
 }  
  
  
 *private void* solve(*double*[][] m) {  
 BigDecimal[][] matrix = Model.*toBigDecimalMatrix*(Model.*cloneMatrix*(m));  
  
 l1 = drawFuncGraph(matrix[0][0], matrix[0][1], matrix[0][2]);  
 l2 = drawFuncGraph(matrix[1][0], matrix[1][1], matrix[1][2]);  
 *if* (hasSingleSolution(Model.*toDoubleMatrix*(matrix))) {  
 intersection = Line.*intersect*(l1, l2);  
 *double* maxCoordinate = getBiggestWorldCoordinate(l1, l2);  
  
 *while* (intersection.getBoundsInParent().isEmpty()) {  
 expandWorld(maxCoordinate);  
 equaliseWorldNStage();  
 l1 = drawFuncGraph(matrix[0][0], matrix[0][1], matrix[0][2]);  
 l2 = drawFuncGraph(matrix[1][0], matrix[1][1], matrix[1][2]);  
 intersection = Line.*intersect*(l1, l2);  
 }  
 screenResult = *new* Coordinates(intersection.getBoundsInParent().getCenterX(), intersection.getBoundsInParent().getCenterY());  
 worldResult = screenToWorld(screenResult);  
 *if* (worldMaxX - Math.*abs*(worldResult.x) < 3 || worldMaxY - Math.*abs*(worldResult.y) < 3) {  
 expandWorld(worldMaxX \* 0.3);  
 equaliseWorldNStage();  
 l1 = drawFuncGraph(matrix[0][0], matrix[0][1], matrix[0][2]);  
 l2 = drawFuncGraph(matrix[1][0], matrix[1][1], matrix[1][2]);  
 intersection = Line.*intersect*(l1, l2);  
 screenResult = *new* Coordinates(intersection.getBoundsInParent().getCenterX(), intersection.getBoundsInParent().getCenterY());  
 worldResult = screenToWorld(screenResult);  
 }  
  
 updateWorld();  
 equaliseWorldNStage();  
 } *else* {  
 *double* maxCoordinate = getBiggestWorldCoordinate(l1, l2);  
 *while* (!isVisibleInCurrSystem(l1) || !isVisibleInCurrSystem(l2)) {  
 expandWorld(maxCoordinate);  
 equaliseWorldNStage();  
 l1 = drawFuncGraph(matrix[0][0], matrix[0][1], matrix[0][2]);  
 l2 = drawFuncGraph(matrix[1][0], matrix[1][1], matrix[1][2]);  
 }  
 updateWorld();  
 }  
  
 l1.setStroke(Color.*BLUE*);  
 l1.setStrokeWidth(2);  
 l2.setStroke(Color.*GREEN*);  
 l2.setStrokeWidth(2);  
  
  
  
 updateWith(l1,l2);  
 *if* (hasSingleSolution(Model.*toDoubleMatrix*(matrix)))  
 designateWorldDot(worldResult);  
 }  
  
 *private void* equaliseWorldNStage() {  
 scaleX = 0.49 \* width / worldMaxX;  
 scaleY = 0.49 \* height / worldMaxY;  
 centrate();  
 }  
  
 *private void* centrate() {  
 offsetX = -width/2/scaleX;  
 offsetY = -height/2/scaleY;  
 }  
  
 *private boolean* isVisibleInCurrSystem(Line l) {  
 *double* visBorderX = worldMaxX \* 0.9;  
 *double* visBorderY = worldMaxY \* 0.9;  
  
 Coordinates screenRectCoord = worldToScreen(-visBorderX, visBorderY);  
 *double* rectScreenWidth = visBorderX \* 2 \* scaleX;  
 *double* rectScreenHeight = visBorderY \* 2 \* scaleY;  
  
 Rectangle visibleBorder = *new* Rectangle(rectScreenWidth,rectScreenHeight);  
 visibleBorder.setX(screenRectCoord.x);  
 visibleBorder.setY(screenRectCoord.y);  
 visibleBorder.setWidth(rectScreenWidth);  
 visibleBorder.setHeight(rectScreenHeight);  
 *return* !Line.*intersect*(visibleBorder, l).getBoundsInParent().isEmpty();  
 }  
  
 *private void* expandWorld(*double* addSize){  
 worldMaxX += addSize;  
 worldMinX -= addSize;  
 worldMaxY += addSize;  
 worldMinY -= addSize;  
 }  
  
 *private boolean* hasSingleSolution(*double*[][] m) {  
 *return* Model.*getMatrixConsistence*(Model.*cloneMatrix*(m)) == 0;  
 }  
  
  
 *private double* getBiggestWorldCoordinate(Line l) {  
 Coordinates s = screenToWorld(l.getStartX(), l.getStartY());  
 Coordinates e = screenToWorld(l.getEndX(), l.getEndY());  
 *double* max = s.x;  
 *double* num;  
 *if*((num = s.y) > max)  
 max = num;  
 *if*((num = e.x) > max)  
 max = num;  
 *if*((num = e.y) > max)  
 max = num;  
 *return* max;  
 }  
  
 *private double* getBiggestWorldCoordinate(Line l1, Line l2) {  
 *double* l1c = getBiggestWorldCoordinate(l1);  
 *double* l2c = getBiggestWorldCoordinate(l2);  
 *return* Math.*max*(l1c, l2c);  
 }  
  
 *private void* designateWorldDot(Coordinates worldDot) {  
 Coordinates horStart = worldToScreen(*new* Coordinates(worldDot.x,0));  
 Coordinates horEnd = worldToScreen(*new* Coordinates(worldDot.x, worldDot.y));  
 Coordinates vertStart = worldToScreen(*new* Coordinates(0, worldDot.y));  
 Coordinates vertEnd = worldToScreen(*new* Coordinates(worldDot.x, worldDot.y));  
 Line horizontal = *new* Line(horStart.x, horStart.y, horEnd.x, horEnd.y);  
 Line vertical = *new* Line(vertStart.x, vertStart.y, vertEnd.x, vertEnd.y);  
 Coordinates xStringLoc = worldToScreen(*new* Coordinates(worldDot.x, 0));  
 Coordinates yStringLoc = worldToScreen(*new* Coordinates(0, worldDot.y));  
 Text xString = *new* Text(Double.*toString*(round(worldDot.x,2)));  
 *double* stringWidth = xString.getLayoutBounds().getWidth();  
 *double* stringHeight = xString.getLayoutBounds().getHeight();  
 xString.setStroke(Color.*RED*);  
 xString.setX(xStringLoc.x - stringWidth/2);  
  
 *if*(worldDot.y > 0)  
 xString.setY(xStringLoc.y + stringHeight);  
 *else* xString.setY(xStringLoc.y - 5);  
  
 *if*(worldDot.x > 0 && worldLengthToScreen(worldMaxX - worldDot.x) <= stringWidth)  
 xString.setX(xStringLoc.x - stringWidth - 3);  
 *else if*(worldDot.x < 0 && worldLengthToScreen(worldMinX - worldDot.x) <= stringWidth)  
 xString.setX(xStringLoc.x + 3);  
  
 Text yString = *new* Text(Double.*toString*(round(worldDot.y,2)));  
 yString.setStroke(Color.*RED*);  
 stringWidth = yString.getLayoutBounds().getWidth();  
 stringHeight = yString.getLayoutBounds().getHeight();  
 *if* (worldDot.x > 0)  
 yString.setX(yStringLoc.x - stringWidth - 5);  
 *else* yString.setX(yStringLoc.x + 3);  
  
 yString.setY(yStringLoc.y + stringHeight /3);  
 *if*(worldDot.y > 0 && worldLengthToScreen(worldMaxY - worldDot.y) <= stringHeight)  
 yString.setY(yStringLoc.y + stringHeight + 3);  
 *else if*(worldDot.y < 0 && worldLengthToScreen(worldMinY - worldDot.y) <= stringHeight)  
 yString.setY(yStringLoc.y - 3);  
  
 horizontal.setStroke(Color.*RED*);  
 vertical.setStroke(Color.*RED*);  
 horizontal.setStrokeWidth(2);  
 vertical.setStrokeWidth(2);  
 worldPane.getChildren().addAll(horizontal, vertical, xString, yString);  
 }  
 *private double* round(*double* value, *int* places) {  
 *if* (places < 0) *throw new* IllegalArgumentException();  
 BigDecimal bd = BigDecimal.*valueOf*(value);  
 bd = bd.setScale(places, RoundingMode.*HALF\_UP*);  
 *return* bd.doubleValue();  
 }  
  
 *private void* prepareWindow() {  
 root = *new* StackPane();  
 scene = *new* Scene(root, width, height);  
 worldPane = *new* Pane();  
 root.getChildren().add(worldPane);  
 stage.setTitle(title);  
 stage.setScene(scene);  
 stage.show();  
 }  
 *private void* update() {  
 updateWorld();  
 solve(slae);  
 }  
  
 *private void* updateWorld() {  
 worldPane.getChildren().clear();  
 drawLines();  
 }  
 *private void* updateWith(Node... n) {  
 worldPane.getChildren().addAll(n);  
 }  
 *private void* drawLines() {  
 *int* smallStrWidth = 1;  
 *int* bigStrWidth = 3;  
 *double* dashSize = worldMaxX/30;  
 *int* gap = (*int*)worldMaxX/10;  
 Line l = *null*;  
 Text xAxisName = *new* Text();  
 Text yAxisName = *new* Text();  
 Text dashNum;  
 xAxisName.setFont(Font.*font*(18));  
 yAxisName.setFont(Font.*font*(18));  
 Coordinates screenS, screenE, axisNameScreenLoc, dashNumLoc;  
 equaliseWorldNStage();  
 *for* (*int* x =(*int*) worldMinX; x <= worldMaxX; x++) {  
 *if* (x != 0 & x%gap == 0) {  
 screenS = worldToScreen(x, -dashSize);  
 screenE = worldToScreen(x, dashSize);  
 l = *new* Line(screenS.x, screenS.y, screenE.x, screenE.y);  
 l.setStrokeWidth(smallStrWidth);  
 dashNum = *new* Text(Integer.*toString*(x));  
 dashNum.setStrokeWidth(1);  
 dashNum.setFont(*new* Font(10));  
 dashNum.setStroke(Color.*GREY*);  
 dashNumLoc = worldToScreen(x - screenLengthToWorld(dashNum.getLayoutBounds().getWidth()/2),  
 -dashSize-screenLengthToWorld(dashNum.getLayoutBounds().getHeight()));  
 dashNum.setX(dashNumLoc.x);  
 dashNum.setY(dashNumLoc.y);  
 worldPane.getChildren().addAll(l, dashNum);  
 } *else if* (x == 0){  
 screenS = worldToScreen(x, worldMinY);  
 screenE = worldToScreen(x, worldMaxY);  
 l = *new* Line(screenS.x, screenS.y, screenE.x, screenE.y);  
 l.setStrokeWidth(bigStrWidth);  
 xAxisName.setText("X");  
 axisNameScreenLoc = worldToScreen(worldMaxX,0);  
 xAxisName.setX(axisNameScreenLoc.x - xAxisName.getLayoutBounds().getWidth() - 3);  
 xAxisName.setY(axisNameScreenLoc.y + xAxisName.getLayoutBounds().getHeight());  
 worldPane.getChildren().add(xAxisName);  
 worldPane.getChildren().add(l);  
 }  
 }  
 *for* (*int* y = (*int*)worldMinY; y <= worldMaxY; y++) {  
 *if* (y != 0 & y%gap == 0) {  
 screenS = worldToScreen(-dashSize,y);  
 screenE = worldToScreen(dashSize, y);  
 l = *new* Line(screenS.x, screenS.y, screenE.x, screenE.y);  
 l.setStrokeWidth(smallStrWidth);  
 dashNum = *new* Text(Integer.*toString*(y));  
 dashNum.setStrokeWidth(1);  
 dashNum.setFont(*new* Font(10));  
 dashNum.setStroke(Color.*GREY*);  
 dashNumLoc = worldToScreen(-dashSize - screenLengthToWorld(dashNum.getLayoutBounds().getWidth()) - 0.3,  
 y - screenLengthToWorld(dashNum.getLayoutBounds().getHeight()/2));  
 dashNum.setX(dashNumLoc.x);  
 dashNum.setY(dashNumLoc.y);  
 worldPane.getChildren().addAll(l, dashNum);  
 } *else if* (y == 0){  
 screenS = worldToScreen(worldMinX, y);  
 screenE = worldToScreen(worldMaxX, y);  
 l = *new* Line(screenS.x, screenS.y, screenE.x, screenE.y);  
 l.setStrokeWidth(bigStrWidth);  
 yAxisName.setText("Y");  
 axisNameScreenLoc = worldToScreen(0, worldMaxY);  
 yAxisName.setX(axisNameScreenLoc.x+4);  
 yAxisName.setY(axisNameScreenLoc.y + yAxisName.getLayoutBounds().getHeight()-5);  
 worldPane.getChildren().add(l);  
 worldPane.getChildren().add(yAxisName);  
  
 }  
  
 }  
  
 }  
  
 *private double* screenLengthToWorld(*double* length) {  
 *return* length / scaleX;  
 }  
  
 *private double* worldLengthToScreen(*double* length) {  
 *return* length \* scaleX;  
 }  
  
 *private* Coordinates worldToScreen(*double* wx, *double* wy) {  
 *return new* Coordinates(  
 worldXToScreen(wx),  
 worldYToScreen(wy)  
 );  
 }  
 *private double* worldYToScreen(*double* wy) {  
 BigDecimal wyBD = BigDecimal.*valueOf*(wy);  
 BigDecimal scaleYBD = BigDecimal.*valueOf*(scaleY);  
 BigDecimal offsetYBD = BigDecimal.*valueOf*(offsetY);  
 *return* wyBD.negate().subtract(offsetYBD).multiply(scaleYBD).doubleValue();  
 } *private double* worldXToScreen(*double* wx) {  
 BigDecimal wxBD = BigDecimal.*valueOf*(wx);  
 BigDecimal scaleXBD = BigDecimal.*valueOf*(scaleX);  
 BigDecimal offsetXBD = BigDecimal.*valueOf*(offsetX);  
 *return* wxBD.subtract(offsetXBD).multiply(scaleXBD).doubleValue();  
 }  
  
 *private* Coordinates worldToScreen(Coordinates world) {  
 *return new* Coordinates(  
 worldXToScreen(world.x),  
 worldYToScreen(world.y)  
 );  
 }  
  
 */\*private double screenXToWorld(double sx) {  
  
 return sx/scaleX + offsetX;  
 }\*/  
 private double* screenXToWorld(*double* sx) {  
 BigDecimal sxBD = BigDecimal.*valueOf*(sx);  
 BigDecimal scaleXBD = BigDecimal.*valueOf*(scaleX);  
 BigDecimal offsetXBD = BigDecimal.*valueOf*(offsetX);  
 *return* sxBD.divide(scaleXBD, RoundingMode.*HALF\_EVEN*).add(offsetXBD).doubleValue();  
 }  
  
 */\*private double screenYToWorld(double sy) {  
 return -(sy/scaleY + offsetY);  
 }\*/  
 private double* screenYToWorld(*double* sy) {  
 BigDecimal syBD = BigDecimal.*valueOf*(sy);  
 BigDecimal scaleYBD = BigDecimal.*valueOf*(scaleY);  
 BigDecimal offsetYBD = BigDecimal.*valueOf*(offsetY);  
 *return* syBD.divide(scaleYBD, RoundingMode.*HALF\_EVEN*).add(offsetYBD).negate().doubleValue();  
 }  
  
 *private* Coordinates screenToWorld(*double* sx, *double* sy) {  
 *return new* Coordinates(  
 screenXToWorld(sx),  
 screenYToWorld(sy)  
 );  
 }  
  
 *private* Coordinates screenToWorld(Coordinates screen) {  
 *return new* Coordinates(  
 screenXToWorld(screen.x),  
 screenYToWorld(screen.y)  
 );  
 }  
 *private* Line drawFuncGraph(BigDecimal xCoef, BigDecimal yCoef, BigDecimal freeCoef){  
 *if*(yCoef.doubleValue() == 0 & xCoef.doubleValue() == 0)  
 *throw new* RuntimeException("Cant draw graph");  
 Coordinates worldStart = *new* Coordinates();  
 Coordinates worldEnd = *new* Coordinates();  
 Coordinates screenStart, screenEnd;  
 BigDecimal wMinYBD = BigDecimal.*valueOf*(worldMinY);  
 BigDecimal wMaxYBD = BigDecimal.*valueOf*(worldMaxY);  
 BigDecimal wMinXBD = BigDecimal.*valueOf*(worldMinX);  
 BigDecimal wMaxXBD = BigDecimal.*valueOf*(worldMaxX);  
 *MathFunctionBD* f = getStraightFunc(xCoef, yCoef, freeCoef);  
  
 *if* (f != *null* && Math.*abs*(getDelta(f)) >= 1) { *//yCoef != 0 & func with xVariable doesnt fit in height size  
 if* ((f = getStraightFunc(yCoef, xCoef, freeCoef)) != *null*) { *//xCoef != 0* worldStart.x = f.count(wMinYBD).doubleValue();  
 worldStart.y = worldMinY;  
 worldEnd.x = f.count(wMaxYBD).doubleValue();  
 worldEnd.y = worldMaxY;  
 } *else* { *//xCoef == 0  
 double* yVal = freeCoef.divide(yCoef).doubleValue();  
 worldStart.x = worldMinX;  
 worldStart.y = yVal;  
 worldEnd.x = worldMaxX;  
 worldEnd.y = yVal;  
 }  
 } *else if*(f != *null*){  
 worldStart.x = worldMinX;  
 worldStart.y = f.count(wMinXBD).doubleValue();  
 worldEnd.x = worldMaxX;  
 worldEnd.y = f.count(wMaxXBD).doubleValue();  
 } *else if*(xCoef.doubleValue() != 0){  
 *double* x = freeCoef.divide(xCoef, 2, RoundingMode.*HALF\_UP*).doubleValue();  
 worldStart.x = x;  
 worldStart.y = worldMinY;  
 worldEnd.x = x;  
 worldEnd.y = worldMaxY;  
 }  
 screenStart = worldToScreen(worldStart);  
 screenEnd = worldToScreen(worldEnd);  
 *return new* Line(screenStart.x, screenStart.y, screenEnd.x, screenEnd.y);  
 }  
 *private MathFunctionBD* getStraightFunc(BigDecimal varCoef, BigDecimal denominator, BigDecimal freeVal){  
 *if* (denominator.doubleValue() == 0)  
 *return null*;  
 *return* var -> (freeVal.subtract(varCoef.multiply(var))).divide(denominator, 2, RoundingMode.*HALF\_UP*);  
 }  
  
  
 *private double* getDelta(*MathFunctionBD* f) {  
 *return* f.count(BigDecimal.*valueOf*(2))  
 .subtract(f.count(BigDecimal.*valueOf*(1)))  
 .doubleValue();  
 }  
  
 *interface MathFunctionBD* {  
 BigDecimal count(BigDecimal num);  
 }  
  
 *private class* Coordinates{  
 *private double* x;  
 *private double* y;  
  
 *private* Coordinates() { }  
  
 *private* Coordinates(*double* x, *double* y) {  
 *this*.x = x;  
 *this*.y = y;  
 }  
  
 *public* String toString() {  
   
 *return* "Coordinates:" + x + " | " + y;  
 }  
 }  
}