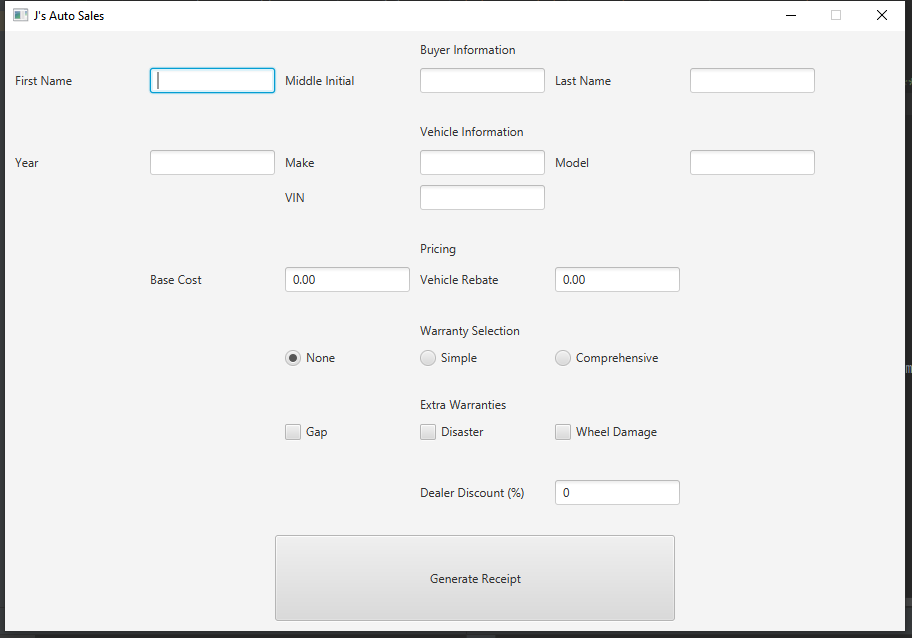
Program Cover Sheet

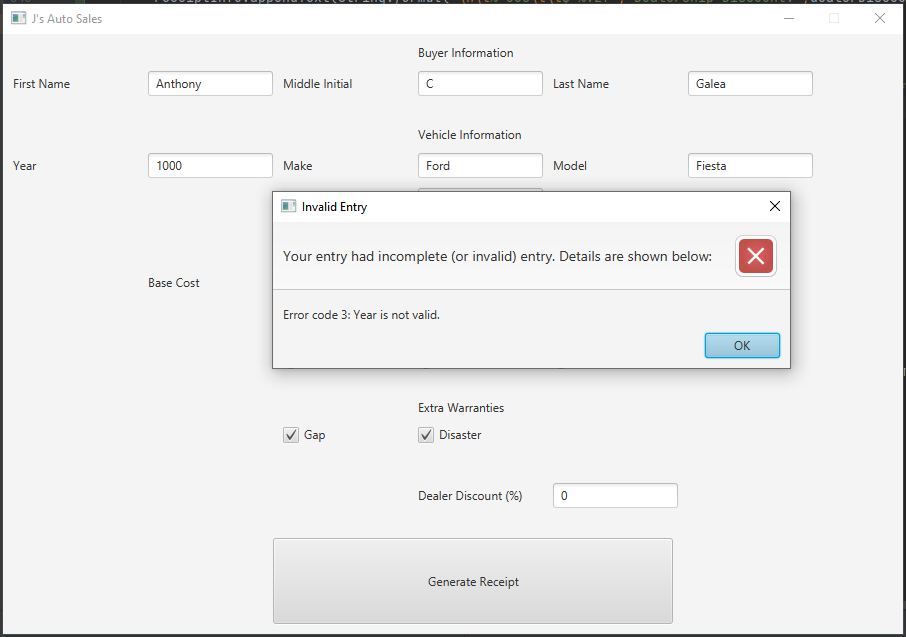
|  |
| --- |
| Name: Anthony Galea |
| Assignment: Assignment 1 |
| List any parts of the assignment that do not work/were not completed: I believe I may have missed some data validation components, such as overflows and validating for certain characters. However, I also believe that names are not only alphanumeric in certain cases, so I did not perform alphanumeric checks on a lot of data.  How to compile and run in command line:  Ensure running Java 8  javac Main.java  java Main |

|  |
| --- |
| Instructor’s Comments: |
| Grade: |

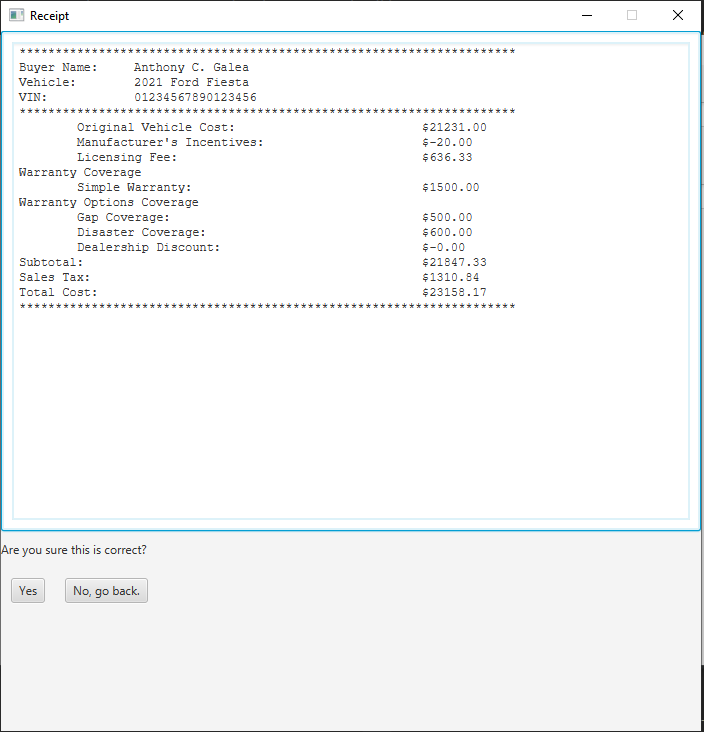
Program Submission Requirements: (1) all files, zipped and uploaded to Canvas and (2) a completed cover sheet, program execution screenshots and source code printed, **stapled** and turned in during class. Failure to follow the submission requirements will result in points lost on that particular assignment.



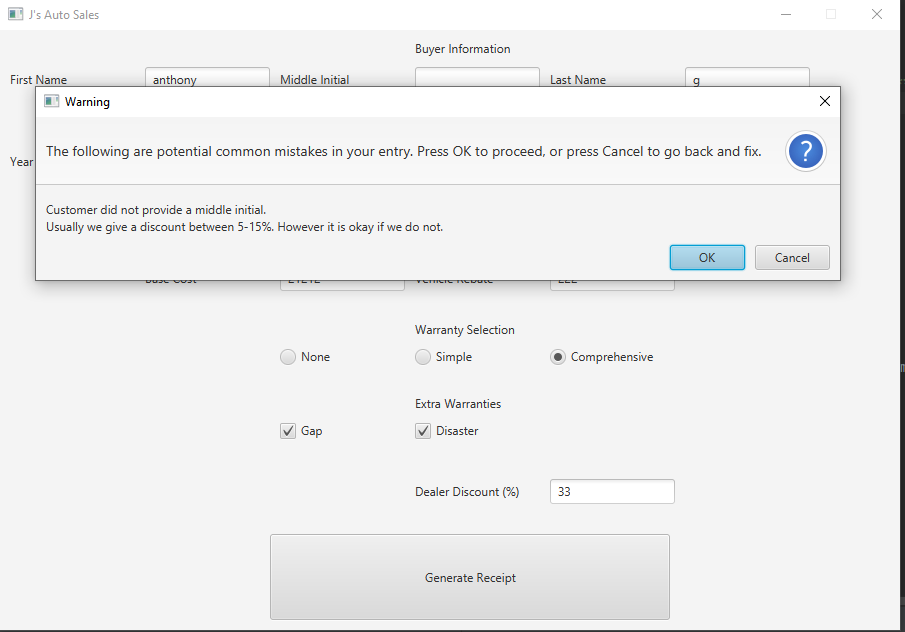
Program on open



Individual invalid data entry error



Receipt created on fully valid data



Warnings that may be thrown by the program.

package com.company;

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.ColumnConstraints;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.HBox;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import java.util.ArrayList;

import java.util.Calendar;

/\*\*

\* This program helps run POS for J's Auto Sales.

\* @author Anthony Galea

\*/

public class Main extends Application {

//Create global variables for text fields and toggles to allow for easy access

TextField fNameInput;

TextField mInitialInput;

TextField lNameInput;

TextField yearInput;

TextField makeInput;

TextField modelInput;

TextField vinInput;

TextField baseCostInput;

TextField rebateInput;

RadioButton noneRadio;

RadioButton simpleRadio;

RadioButton compRadio;

CheckBox gapCoverage;

CheckBox disasterCoverage;

CheckBox wheelCoverage;

TextField dealerDiscountInput;

final int windowWidth = 900;

final int windowHeight = 600;

public static void main(String[] args) {

// Launch JavaFX Code

launch(args);

}

/\*\*

\* Sets spacing properly for a given pane to standardize pane spacing.

\* @param pane The pane needing to be given the spacing

\*/

private void setGridPaneSpacing(GridPane pane) {

pane.setVgap(10);

pane.setHgap(10);

ColumnConstraints colConst = new ColumnConstraints();

colConst.setPrefWidth(125);

pane.setPadding(new Insets(10, 10, 10, 10));

pane.getColumnConstraints().addAll(colConst, colConst, colConst, colConst, colConst, colConst);

}

/\*\*

\* Creates a Graphical User Interface with all necessary information about the buyer being collected.

\* @return Returns a GridPane with the buyer GUI objects

\*/

public GridPane createBuyerGUI() {

GridPane buyerGridPane = new GridPane(); //For buyer information

fNameInput = new TextField();

mInitialInput = new TextField();

lNameInput = new TextField();

buyerGridPane.add(new Label("Buyer Information"), 3, 0);

buyerGridPane.add(new Label("First Name"), 0, 1);

buyerGridPane.add(fNameInput, 1, 1);

buyerGridPane.add(new Label("Middle Initial"), 2, 1);

buyerGridPane.add(mInitialInput, 3, 1);

buyerGridPane.add(new Label("Last Name"), 4, 1);

buyerGridPane.add(lNameInput, 5, 1);

buyerGridPane.setPadding(new Insets(10, 10, 10, 10));

setGridPaneSpacing(buyerGridPane);

return buyerGridPane;

}

/\*\*

\* Creates a Graphical User Interface with all necessary information about the vehicle being collected.

\* @return Returns a GridPane with the vehicle GUI objects

\*/

public GridPane createVehicleGUI() {

GridPane vehicleGridPane = new GridPane(); //For vehicle information

yearInput = new TextField();

makeInput = new TextField();

modelInput = new TextField();

vinInput = new TextField();

vehicleGridPane.add(new Label("Vehicle Information"), 3, 0);

vehicleGridPane.add(new Label("Year"), 0, 1);

vehicleGridPane.add(yearInput, 1, 1);

vehicleGridPane.add(new Label("Make"), 2, 1);

vehicleGridPane.add(makeInput, 3, 1);

vehicleGridPane.add(new Label("Model"), 4, 1);

vehicleGridPane.add(modelInput, 5, 1);

vehicleGridPane.add(new Label("VIN"), 2, 2);

vehicleGridPane.add(vinInput, 3, 2);

setGridPaneSpacing(vehicleGridPane);

return vehicleGridPane;

}

/\*\*

\* Creates a Graphical User Interface with all necessary information about the pricing of the vehicle being sold.

\* @return Returns a GridPane with the vehicle pricing GUI objects

\*/

public GridPane createPricingGUI() {

GridPane pricingGridPane = new GridPane(); //For base pricing information

baseCostInput = new TextField();

baseCostInput.setText("0.00");

rebateInput = new TextField();

rebateInput.setText("0.00");

pricingGridPane.add(new Label("Pricing"), 3, 0);

pricingGridPane.add(new Label("Base Cost"), 1, 1);

pricingGridPane.add(baseCostInput, 2, 1);

pricingGridPane.add(new Label("Vehicle Rebate"), 3, 1);

pricingGridPane.add(rebateInput, 4, 1);

pricingGridPane.setPadding(new Insets(10, 10, 10, 10));

setGridPaneSpacing(pricingGridPane);

return pricingGridPane;

}

/\*\*

\* Function to handle the value changing in the radio buttons for warranty coverage options.

\* @param selection Input of the radio button being selected

\*/

public void radioHandler(int selection) {

noneRadio.setSelected(false);

simpleRadio.setSelected(false);

compRadio.setSelected(false);

switch(selection) {

case 0:

noneRadio.setSelected(true);

wheelCoverage.setVisible(true);

break;

case 1:

simpleRadio.setSelected(true);

wheelCoverage.setVisible(false);

wheelCoverage.setSelected(false);

break;

default:

compRadio.setSelected(true);

wheelCoverage.setVisible(false);

wheelCoverage.setSelected(false);

break;

}

}

/\*\*

\* Creates a Graphical User Interface with all necessary information about the warranty being sold to cover the vehicle.

\* @return Returns a GridPane with the warranty selection GUI objects

\*/

public GridPane createWarrantyRadios() {

GridPane warrantyGridPane = new GridPane(); //For selecting a base warranty options

noneRadio = new RadioButton("None");

noneRadio.setSelected(true); //Set to None by default

noneRadio.setOnAction(event -> radioHandler(0));

simpleRadio = new RadioButton("Simple");

simpleRadio.setSelected(false);

simpleRadio.setOnAction(event -> radioHandler(1));

compRadio = new RadioButton("Comprehensive");

compRadio.setSelected(false);

compRadio.setOnAction(event -> radioHandler(2));

warrantyGridPane.add(new Label("Warranty Selection"), 3, 0);

warrantyGridPane.add(noneRadio, 2, 1);

warrantyGridPane.add(simpleRadio, 3, 1);

warrantyGridPane.add(compRadio, 4, 1);

setGridPaneSpacing(warrantyGridPane);

return warrantyGridPane;

}

/\*\*

\* Creates a Graphical User Interface with all necessary information about the extra warranty objects being sold to cover the vehicle.

\* @return Returns a GridPane with the extra warranty option selection GUI objects

\*/

public GridPane createExtraWarrantySelects() {

GridPane extraWarranty = new GridPane();

gapCoverage = new CheckBox("Gap");

disasterCoverage = new CheckBox("Disaster");

wheelCoverage = new CheckBox("Wheel Damage");

extraWarranty.add(new Label("Extra Warranties"), 3, 0);

extraWarranty.add(gapCoverage,2,1);

extraWarranty.add(disasterCoverage,3,1);

extraWarranty.add(wheelCoverage,4,1);

setGridPaneSpacing(extraWarranty);

return extraWarranty;

}

/\*\*

\* Creates a Graphical User Interface with all necessary information about the dealer discount being sold to cover the vehicle.

\* @return Returns a GridPane with the dealer discount GUI objects

\*/

public GridPane createDealerDiscount() {

GridPane dealerPane = new GridPane();

dealerDiscountInput = new TextField();

dealerDiscountInput.setText("0");

dealerPane.add(new Label("Dealer Discount (%)"), 3, 1);

dealerPane.add(dealerDiscountInput,4,1);

setGridPaneSpacing(dealerPane);

return dealerPane;

}

/\*\*

\* Creates a GridPane for the button that calculates the total sale.

\* @return Returns a GridPane housing the generate receipt button

\*/

public GridPane generateReceiptButton() {

GridPane receiptBtnPane = new GridPane();

Button receiptBtn = new Button("Generate Receipt");

receiptBtn.setOnAction(event -> {

if (checkForErrorsAndWarnings()) {

printReceipt();

}

});

receiptBtn.setPrefWidth(400);

receiptBtn.setPrefHeight(100);

receiptBtnPane.add(receiptBtn, 1, 0);

setGridPaneSpacing(receiptBtnPane);

ColumnConstraints colConst1 = new ColumnConstraints(); //For column 1, overriding the column widths from the built-in function

colConst1.setPrefWidth(250);

ColumnConstraints colConstBtn = new ColumnConstraints(); //For the column with the button, overriding the column widths from the built in function

colConstBtn.setPrefWidth(400);

receiptBtnPane.getColumnConstraints().clear();

receiptBtnPane.getColumnConstraints().addAll(colConst1, colConstBtn);

return receiptBtnPane;

}

/\*\*

\* Performs data validation on all user entry fields.

\* @return Returns true if there are no errors and the user chooses to ignore all (if applicable) warnings. False otherwise.

\*/

public boolean checkForErrorsAndWarnings() {

ArrayList<Integer> errors = new ArrayList<>();

ArrayList<Integer> warnings = new ArrayList<>();

//Check for valid first name

if (fNameInput.getText().length() == 0) {

errors.add(0); //error code 0: first name empty

}

//Check for valid middle initial

if (mInitialInput.getText().length() > 1) {

errors.add(1); //error code 1: too long middle initial entry

} else if (mInitialInput.getText().length()==0) {

warnings.add(0); //Some people have no middle names, so it is not breaking, however it is something to check on.

}

//Check for valid last name

if (lNameInput.getText().length() == 0) {

errors.add(2); //error code 2: last name error

}

//Check for valid vehicle year (first car was made in 1886)

int year = -1; //For validation on VIN

try {

year = Integer.parseInt(yearInput.getText());

if (!(year >=1886 && year <= Calendar.getInstance().get(Calendar.YEAR)+1)) { //Usually car models are taken the year after the current year, so this should evolve over time.

errors.add(3); //error code 3: year error

}

} catch (Exception e) {

errors.add(3); //error code 3: year error

}

//Check for valid make

if (makeInput.getText().length() == 0) {

errors.add(4); //error code 4: no make provided

}

//Check for valid model

if (modelInput.getText().length() == 0) {

errors.add(5); //Error code 5: no model provided

}

//Check for valid price

double numPrice = -1; //For data validation on rebate

try {

numPrice = Double.parseDouble(baseCostInput.getText());

String price = baseCostInput.getText();

if (price.contains(".")) {

if(baseCostInput.getText().indexOf(".") < baseCostInput.getText().length()-3) {

errors.add(6); //Error code 6: no valid base price provided

}

}

} catch(Exception e) {

errors.add(6); //Error code 6: no valid base price provided

}

//Check for valid rebate price

double rebate = -1;

try {

String rebateText = rebateInput.getText();

rebate = Double.parseDouble(rebateInput.getText());

if (rebateText.contains(".")) {

if(rebateText.indexOf(".") < rebateText.length()-3) {

errors.add(7); //Error code 7: no valid rebate price provided

}

}

} catch (Exception e) {

//e.printStackTrace();

errors.add(7);

}

if ((numPrice != -1 && numPrice < rebate) ||rebate < 0) {

errors.add(7); //Error code 7: invalid rebate price

}

//check for valid VIN

String vin = vinInput.getText();

if (vin.length()==17 && year >= 1981) { //VIN standard came about in 1981

boolean valid = true;

for (int i = 0; i < vin.length(); i++) {

//VIN standard is 17 alphanumeric characters outside of O, I, and Q

if (!"ABCDEFGHJKLMNPRSTUVWXYZabcdefghjklmnprstuvwxyz0123456789".contains(""+vin.charAt(i))) {

valid = false;

break;

}

}

if (!valid) {

errors.add(8); //Error code 8: Invalid VIN

}

} else if (year < 1981) {

boolean valid = true;

for (int i = 0; i < vin.length(); i++) {

//VIN standard is 17 alphanumeric characters outside of O, I, and Q

if (!"ABCDEFGHJKLMNPRSTUVWXYZabcdefghjklmnprstuvwxyz0123456789IOQioq".contains(""+vin.charAt(i))) {

valid = false;

break;

}

}

if (!valid) {

errors.add(8); //Error code 8: Invalid VIN

}

} else {

errors.add(8); //Error code 8: Invalid VIN

}

try {

double discount = Double.parseDouble(dealerDiscountInput.getText());

if (discount >=100 || discount < 0) {

errors.add(9);

}

if ((discount < 5 || discount > 15)&&discount!=0) {//Discounts are optional, however they are usually between 5 and 15%, if applicable.

warnings.add(1);

}

} catch (Exception e) {

errors.add(9); //Error code 9: invalid discount

}

if (errors.size() > 0) {

failed(errors);

return false; //return false if errors exist

}

if (warnings.size() > 0) {

return displayWarnings(warnings);

}

return true;

}

/\*\*

\* Creates an Alert to confirm that the user intends to send the information that may be incorrectly validated.

\* @param warnings ArrayList containing info on which warnings to display to the user.

\* @return Returns whether the user wishes to ignore the warnings (true) or not (false).

\*/

public boolean displayWarnings(ArrayList<Integer> warnings) {

Alert alert = new Alert(Alert.AlertType.CONFIRMATION);

alert.setTitle("Warning");

alert.setHeaderText("The following are potential common mistakes in your entry. Press OK to proceed, or press Cancel to go back and fix.");

String warningMessage = "";

if (warnings.contains(0)) {

warningMessage+="Customer did not provide a middle initial.\n";

}

if (warnings.contains(1)) {

warningMessage+="Usually we give a discount between 5-15%. However it is okay if we do not.";

}

alert.setContentText(warningMessage);

alert.showAndWait();

//User will confirm or deny.

return alert.getResult() ==ButtonType.OK;

}

/\*\*

\* Creates an Alert informing the user that their data was not fully validated.

\* @param errors ArrayList containing info on which errors to display to the user.

\*/

public void failed(ArrayList<Integer> errors) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Invalid Entry");

alert.setHeaderText("Your entry had incomplete (or invalid) entry. Details are shown below:");

alert.setResizable(false);

String errorMessage = "";

if (errors.contains(0)) {

errorMessage+="Error code 0: First Name entry is empty.\n";

}

if (errors.contains(1)) {

errorMessage+="Error code 1: Middle Initial is too long.\n";

}

if (errors.contains(2)) {

errorMessage+="Error code 2: Last name entry is empty.\n";

}

if (errors.contains(3)) {

errorMessage+="Error code 3: Year is not valid.\n";

}

if (errors.contains(4)) {

errorMessage+="Error code 4: No make provided.\n";

}

if (errors.contains(5)) {

errorMessage+="Error code 5: No model provided.\n";

}

if (errors.contains(6)) {

errorMessage+="Error code 6: No valid base price provided.\n";

}

if (errors.contains(7)) {

errorMessage+="Error code 7: Invalid rebate price input.\n";

}

if (errors.contains(8)) {

errorMessage+="Error code 8: VIN input is invalid.\n";

}

if (errors.contains(9)) {

errorMessage+="Error code 9: Discount is invalid.\n";

}

alert.setContentText(errorMessage);

alert.showAndWait();

}

/\*\*

\* Surveys the existing state of the radio buttons for warranty selection and returns the cost of warranty in excess to the vehicle cost.

\* @return Returns the cost of warranty for the vehicle.

\*/

public double getWarrantyCost() {

if (noneRadio.isSelected()) {

return 0;

} else if (simpleRadio.isSelected()) {

return 1500;

} else {

return 2750;

}

}

/\*\*

\* Surveys the existing state of the radio buttons for warranty selection and returns the name of warranty.

\* @return Returns the name of the warranty plan selected.

\*/

public String getWarrantyName() {

if (noneRadio.isSelected()) {

return "No Warranty:\t";

} else if (simpleRadio.isSelected()) {

return "Simple Warranty:\t";

} else {

return "Comprehensive Warranty:\t";

}

}

/\*\*

\* Gets how many selection boxes are chosen.

\* @return Amount of option warranties are selected.

\*/

public int getOptionsSelected() {

int selected = 0;

if (gapCoverage.isSelected()) {

selected++;

}

if (disasterCoverage.isSelected()) {

selected++;

}

if (wheelCoverage.isSelected()) {

selected++;

}

return selected;

}

/\*\*

\* Adds lines to receipt of option warranty coverage.

\* @return String of lines to add to receipt

\*/

public String optionsForReceipt() {

String result = "";

if (getOptionsSelected()==0) {

return "\tNo Options Selected";

}

if (gapCoverage.isSelected()) {

result+=String.format("\n\t%-26s\t\t\t$%.2f","Gap Coverage:",500.00);

}

if (disasterCoverage.isSelected()) {

result+=String.format("\n\t%-26s\t\t\t$%.2f","Disaster Coverage:",600.00);

}

if (wheelCoverage.isSelected()) {

result+=String.format("\n\t%-26s\t\t\t$%.2f","Wheel Damage Coverage:",300.00);

}

return result;

}

/\*\*

\* Resets form for a new customer.

\*/

public void clearForm() {

fNameInput.clear();

mInitialInput.clear();

lNameInput.clear();

yearInput.clear();

makeInput.clear();

modelInput.clear();

vinInput.clear();

baseCostInput.setText("0.00");

rebateInput.setText("0.00");

gapCoverage.setSelected(false);

disasterCoverage.setSelected(false);

wheelCoverage.setSelected(false);

dealerDiscountInput.setText("0");

radioHandler(0);

}

/\*\*

\* Creates new window for displaying the receipt, and displays the receipt on it for confirmation.

\*/

public void printReceipt() {

Stage receiptStage = new Stage();

receiptStage.setTitle("Receipt");

VBox root = new VBox();

HBox confirm = new HBox();

confirm.setSpacing(20);

root.setSpacing(10);

double originalCost = Double.parseDouble(baseCostInput.getText()); //try and catch are done during the error checking,

double incentive = Double.parseDouble(rebateInput.getText());

double vehicleCost = originalCost-incentive;

double licenseFee = vehicleCost\*0.03;

double warrantyFees=0;

double warrantyOptions=0;

double dealerDiscount = Double.parseDouble(dealerDiscountInput.getText());

double subtotal = vehicleCost+licenseFee+warrantyFees+warrantyOptions-dealerDiscount;

double salesTax = subtotal\*0.06;

double totalCost = subtotal+salesTax;

String buyerName = fNameInput.getText() + " ";

if (mInitialInput.getText().length() == 1) {

buyerName += mInitialInput.getText()+". ";

}

buyerName += lNameInput.getText();

String vehicleInformation = yearInput.getText() + " " + makeInput.getText() + " " + modelInput.getText();

String vin = vinInput.getText();

TextArea receiptInfo = new TextArea();

receiptInfo.setStyle("-fx-font-family: 'monospaced';");

receiptInfo.setEditable(false); //Set to read only

receiptInfo.clear();

receiptInfo.setPrefHeight(500);

receiptInfo.appendText("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

receiptInfo.appendText(String.format("\n%-15s\t%s","Buyer Name:",buyerName));

receiptInfo.appendText(String.format("\n%-15s\t%s","Vehicle:",vehicleInformation));

receiptInfo.appendText(String.format("\n%-15s\t%s","VIN:",vin));

receiptInfo.appendText("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

receiptInfo.appendText(String.format("\n\t%-36s\t\t$%.2f","Original Vehicle Cost:",originalCost));

receiptInfo.appendText(String.format("\n\t%-36s\t\t$-%.2f","Manufacturer's Incentives:",incentive));

receiptInfo.appendText(String.format("\n\t%-36s\t\t$%.2f","Licensing Fee:",licenseFee));

receiptInfo.appendText(String.format("\n%s","Warranty Coverage"));

receiptInfo.appendText(String.format("\n\t%-36s\t$%.2f",getWarrantyName(),getWarrantyCost()));

receiptInfo.appendText(String.format("\n%s","Warranty Options Coverage"));

receiptInfo.appendText(optionsForReceipt());

receiptInfo.appendText(String.format("\n\t%-36s\t\t$-%.2f","Dealership Discount:",dealerDiscount));

receiptInfo.appendText(String.format("\n%-36s\t\t\t$%.2f","Subtotal:",subtotal));

receiptInfo.appendText(String.format("\n%-36s\t\t\t$%.2f","Sales Tax:",salesTax));

receiptInfo.appendText(String.format("\n%-36s\t\t\t$%.2f","Total Cost:",totalCost));

receiptInfo.appendText("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

Button confirmData = new Button("Yes");

confirmData.setOnAction(e-> {

receiptStage.close();

clearForm();

});

Button denyData = new Button("No, go back.");

denyData.setOnAction(e-> { //Go back to data entry to fix issues

receiptStage.close();

});

confirm.getChildren().addAll(confirmData, denyData);

receiptInfo.setPadding(new Insets(10, 10, 10, 10));

confirm.setPadding(new Insets(10, 10, 10, 10));

root.getChildren().addAll(receiptInfo, new Label("Are you sure this is correct?"), confirm);

receiptStage.setScene(new Scene(root, 700, 700));

receiptStage.setResizable(false);

receiptStage.show();

}

public void start(Stage primaryStage) {

primaryStage.setTitle("J's Auto Sales");

//Create root vbox for housing all GridPanes

VBox root=new VBox();

root.setSpacing(10);

//Add all elements to VBox

root.getChildren().addAll(createBuyerGUI(), createVehicleGUI(), createPricingGUI(), createWarrantyRadios(), createExtraWarrantySelects(),createDealerDiscount(), generateReceiptButton());

primaryStage.setScene(new Scene(root, windowWidth, windowHeight));

primaryStage.setResizable(false);

primaryStage.show();

}

}