**Advanced Programming Concepts Coursework Report 2015**

**Introduction**

The task at hand is to design an order system for the company PipesR’us. The order system should allow the customer to place orders of pipes with multiple options. The project is to deliver a robust ordering system that would allow users to place as many orders for pipes as they wished in one session, the system should also allow users to add as many types of pipes to the order as they wish and if the company cannot supply the amount the customer would like it should tell the customer this.

**PipesR’us ordering system**

When the team was given the specification the first thing discussed was the interaction we wanted to product to have with the customer. Creating an order system that would limit the users input on the order panel itself so no incompatible pipes could be ordered. This could be done by restricting the options available to the customer as soon as the order panel is opened. At first the customer must select a grade option, if they do not do this and try to submit an order they will be prompted with an error message asking them to select a grade they would like for the pipe. The next customer limitation made was to make sure they had selected how many colours they would like the pipe to have, if the user did not specify this they would them be prompted again with another error message asking them to select how many colours they would like. After this has been done the user then has the option to add extras and as stated before the user would have to select the correct grade to gain access to the potential options for example a grade five pipe must have 2 colours and can have all of the extras but for the customer to be able to select outer reinforcement they pipe must have inner insulation selected first. If the user decides they do not want any of these extras they can then go straight to selecting the size and quantity, again if the user incorrect the wrong input they will be prompted with error messages telling them to enter the correct format. Once the user has finished making their choice for the pipe they can select add to cart to place the pipe into the order system, they can add as many pipes as they would like, as long as PipesR’us can supply them, the user will be told if they cannot. When then user has finished their order they can then decide if they would like to place a new order or check out if the customer decides to place a new order they can do so and there current order total will be stored in the checkout order total. Once all orders have been finished the user can check out and be prompted with the price for the order.

**Assumptions and limitations**

The first assumption made is the number of possible types of pipes the company can supply the customer. For this we had to determine what characteristic each type of pipe could have. From the given graph the first thing noticed is that all pipes work on a plastic grading system, these are graded from one to five. From this you can determine the type of pipe that could be created based on the other options available for pipes such as the number of colours, inner insulation and outer reinforcement. From this the next assumption we had to make is that the colours a pipe could have determined the next stage in checking the pipe type, for example if the pipe had zero, one or two colours. The way our product checks for which type of pipe the customer is ordering is done from our colour options selections on the order panel. The customer can choose from three different options. If the customer chooses the zero colour option the pipe type can only be 1, if the customer selects one colour it can only be type two. The first limitation appeared here as we had to think about what would happen if the customer selects two colours. The product would have to realise that there are three types of pipe that can have this option and then determine which type of pipe the customer had chosen from the selection given. To do this the options a customer can select have been limited on the order panel to restrict errors from occurring when orders of pipes are being made and to keep it as simple as possible for the customer as it is essential that they have a good experience while placing their order. This would give the customer a less irritating experience than being given an error message every time they order an incompatible pipe. Once all of these conditions were met the pipe type could be determined very easily as to be a pipe of type three the user would have to select a pipe of grade 3 to five with at least two colours. For the customer to order a pipe of type 4 they would have to select a grade of three to five, two colours and inner insulation and for a pipe of type five all of the above plus outer reinforcement.

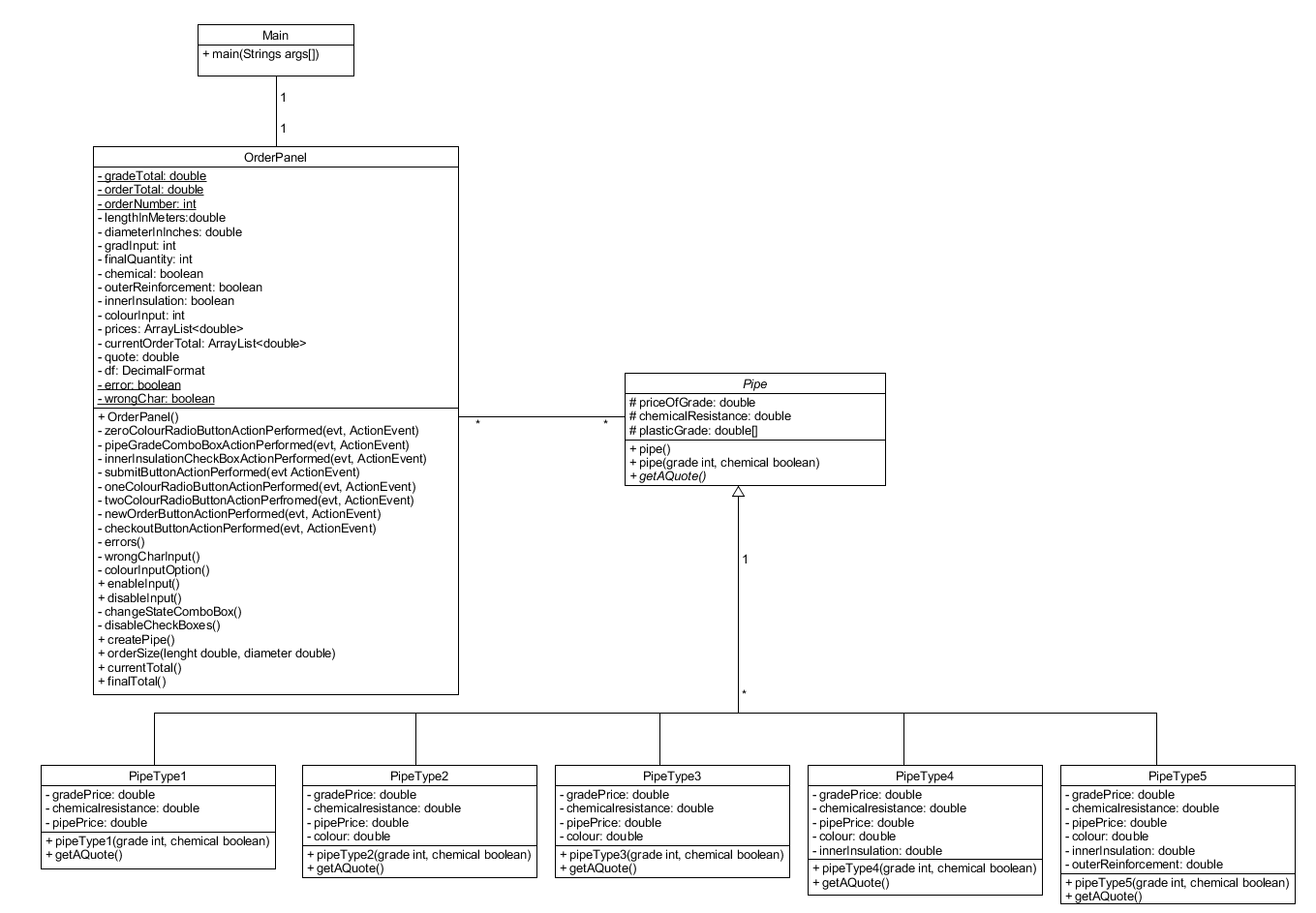
The next assumption also came with limitations as PipesR’us stated that they could only supply pipes up to six meters in length, this meant that in our product we would have to make sure the user would be prompted if with an appropriate error message to state that they have entered an incorrect length specified, the program will also check for incorrect character inputs. Next came the diameter of the product no requirements were given, after some serious thought we decided that the maximum diameter for a pipe should be thirty inches, this seemed like a realistic allowance for a company to specify. When a customer is ordering they will be asked to specify a diameter in inches and if this input exceeds the restriction they will be prompted with an error.

**Testing Plan**

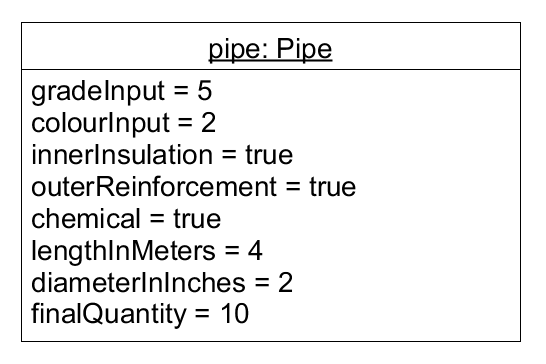
When thinking of possible tests for the product things that came to mind to test whether or not the pipe order system allows the user to order every type of pipe that the company has on offer with prices. This was an easy test to do but quite lengthy, screenshots for this are in the appendix. Testing error messages work correctly and are robust enough for the user to understand the issue that has been created. First Error to test is to see if all the compatibility tests are working correctly for example making sure the user limitations put in place by the team are working correctly, when the user opens the panel everything should be disabled until a pipe grade is selected and if the pipe grade is not selected the error message please select the grade you would like for your pipe should appear in the error message box should be please enter a grade if you just hit submit. Once the grade is selected the next test is to see if the correct error message prints out if the user does not select any colours. Following on from this if the user enters the wrong format for our length diameter and quantity boxes the user should be prompted with an appropriate error message asking them the correct this. This can be tested by entering numbers that exceed the set boundaries of the program in our case this would be seven or higher for length, thirty one for diameter and three hundred for quantity. Other checks to be taken here are to enter minus numbers symbols and a mixture of numbers letters and symbols. Checking if the order system correctly works out the price of pipes and can add multiple pipes and orders for the customer before checking out. a few screen shots are placed here see appendix for the rest of the testing screenshots. Add screen shots and evaluation.

**UML Diagrams (Maybe put this one in a landscape slide)**

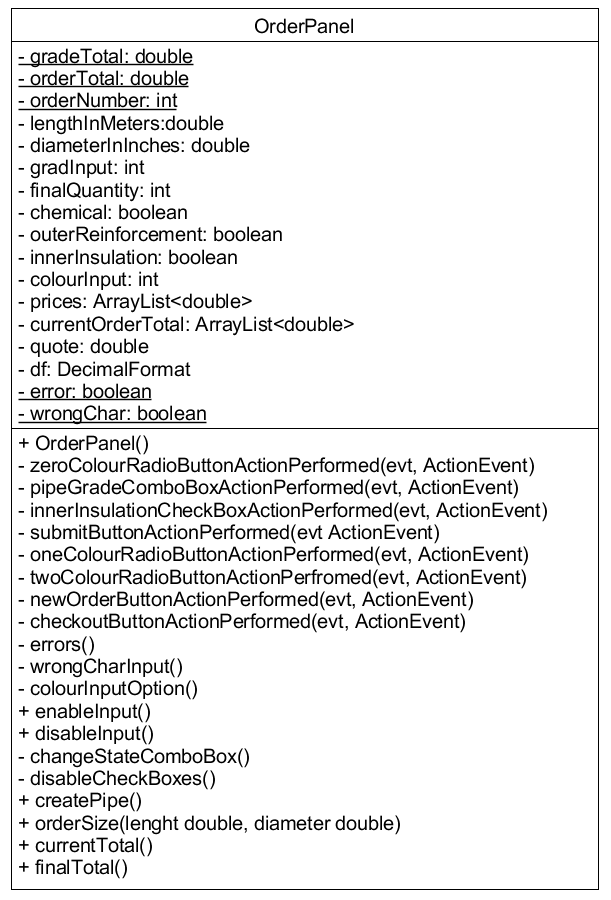
**Class Hierarchy**



**Instance Diagram**



**Class Diagram**



**UML Use Case Diagram** (Check if this is correct)

**Appendix**

**Code CHECK TO MAKE SURE IT’S THE SAME AS WHAT WE HANED IN**

**/\*\***

**\* Welcome to the PipeR'us ordering system this is the main program used**

**\* to initialise our order panel for the customer to use.**

**\***

**\* @author up623013**

**\* @author up674771**

**\* @version 01/12/2015**

**\*/**

**package pipesr.us;**

**/\*\***

**\***

**\* @author unipc**

**\*/**

**public class Main {**

**/\*\***

**\* @param args the command line arguments**

**\*/**

**public static void main(String args[]) {**

**/\* Set the Nimbus look and feel \*/**

**//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">**

**/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.**

**\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html**

**\*/**

**try {**

**for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {**

**if ("Nimbus".equals(info.getName())) {**

**javax.swing.UIManager.setLookAndFeel(info.getClassName());**

**break;**

**}**

**}**

**} catch (ClassNotFoundException ex) {**

**java.util.logging.Logger.getLogger(OrderPanel.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);**

**} catch (InstantiationException ex) {**

**java.util.logging.Logger.getLogger(OrderPanel.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);**

**} catch (IllegalAccessException ex) {**

**java.util.logging.Logger.getLogger(OrderPanel.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);**

**} catch (javax.swing.UnsupportedLookAndFeelException ex) {**

**java.util.logging.Logger.getLogger(OrderPanel.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);**

**}**

**//</editor-fold>**

**/\* Create and display the form \*/**

**java.awt.EventQueue.invokeLater(new Runnable() {**

**public void run() {**

**new OrderPanel().setVisible(true);**

**}**

**});**

**}**

**}**

**package pipesr.us;**

**import java.text.DecimalFormat;**

**import java.util.ArrayList;**

**/\*\***

**\* This is a class for PipesR'us that is used to create the order panel for the**

**\* PipesR'us ordering system, The panel gives the user a range of options to**

**\* choose from when creating a pipe then tells the user if it is possible to**

**\* make said pipe, The class also returns the cost of the order to the user and**

**\* allows the user to create multiple orders.**

**\***

**\* @author up623013**

**\* @author up674771**

**\* @version 01/12/2015**

**\*/**

**public class OrderPanel extends javax.swing.JFrame {**

**private static double grandTotal;**

**private static double orderTotal;**

**private static int orderNumber = 1;**

**private double lengthInMeters;**

**private double diameterInInches;**

**private int gradeInput;**

**private int finalQuantity;**

**private boolean chemical = false;**

**private boolean outerReinforcement = false;**

**private boolean innerInsulation = false;**

**private int colourInput = 0;**

**private final ArrayList<Double> prices = new ArrayList<>();**

**private final ArrayList<Double> currentOrderTotal = new ArrayList<>();**

**private double quote;**

**private final DecimalFormat df = new DecimalFormat("#.##");**

**private static boolean error = false;**

**private static boolean wrongChar = false;**

**/\*\***

**\* Creates new form OrderPanel**

**\*/**

**public OrderPanel() {**

**initComponents();**

**this.changeStateComboBox();**

**chemicalResistanceCheckBox.setEnabled(false);**

**}**

**/\*\***

**\* An ActionPerformed method to check if the colour option "0 colours" is**

**\* selected and then restrict or enable the user to select the extra options**

**\* they want.**

**\***

**\* @param evt used to tell the action listener an action has been performed.**

**\* on the radio buttons**

**\*/**

**private void zeroColourRadioButtonActionPerformed(java.awt.event.ActionEvent evt) {**

**this.colourOptionInput();**

**}**

**/\*\***

**\* A method to add an action event to the JComboBox once a value is selected**

**\* the actions stated below will be performed.**

**\***

**\* @param gradeAction the actions to be preformed once the value in the**

**\* combo box has been selected.**

**\*/**

**private void pipeGradeComboBoxActionPerformed(java.awt.event.ActionEvent evt) {**

**Object inputGrade = pipeGradeComboBox.getSelectedItem();**

**gradeInput = Integer.parseInt((String) inputGrade);**

**switch (gradeInput) {**

**case 0:**

**chemicalResistanceCheckBox.setSelected(false);**

**this.changeStateComboBox();**

**break;**

**case 1:**

**this.changeStateComboBox();**

**zeroColourRadioButton.setEnabled(true);**

**break;**

**case 2:**

**this.changeStateComboBox();**

**zeroColourRadioButton.setEnabled(true);**

**oneColourRadioButton.setEnabled(true);**

**twoColourRadioButton.setEnabled(true);**

**break;**

**case 3:**

**this.changeStateComboBox();**

**zeroColourRadioButton.setEnabled(true);**

**oneColourRadioButton.setEnabled(true);**

**twoColourRadioButton.setEnabled(true);**

**break;**

**case 4:**

**this.changeStateComboBox();**

**oneColourRadioButton.setEnabled(true);**

**twoColourRadioButton.setEnabled(true);**

**break;**

**case 5:**

**this.changeStateComboBox();**

**twoColourRadioButton.setEnabled(true);**

**break;**

**}**

**}**

**/\*\***

**\* A method to test if inner Insulation has been selected and if so to**

**\* enable the outer reinforcement checkbox if the grade selected is 2 or**

**\* above. also deselects outer reinforcement if inner insulation is not**

**\* selected.**

**\***

**\* @param evt used to see if an action has been performed on the inner**

**\* insulation checkbox.**

**\*/**

**private void innerInsulationCheckBoxActionPerformed(java.awt.event.ActionEvent evt) {**

**if (gradeInput == 2) {**

**outerReinforcementCheckBox.setEnabled(false);**

**} else {**

**outerReinforcementCheckBox.setEnabled(true);**

**}**

**if (innerInsulationCheckBox.isSelected() == false) {**

**outerReinforcementCheckBox.setSelected(false);**

**}**

**}**

**/\*\***

**\* Tests all input fields to make sure they are entering numbers, takes in a**

**\* string and converts it to a double or an int.**

**\***

**\*/**

**private void submitButtonActionPerformed(java.awt.event.ActionEvent evt) {**

**errorTextArea.setText(null);**

**try {**

**String stringLength = lengthTextField.getText();**

**lengthInMeters = Double.parseDouble(stringLength);**

**String stringDiameter = diameterTextField.getText();**

**diameterInInches = Double.parseDouble(stringDiameter);**

**String quantity = quantityTextField.getText();**

**finalQuantity = Integer.parseInt(quantity);**

**this.errors();**

**if (error == false) {**

**/\*\***

**\* Checks which check boxes and radio buttons are selected and**

**\* returns true.**

**\***

**\*/**

**chemical = chemicalResistanceCheckBox.isSelected();**

**outerReinforcement = outerReinforcementCheckBox.isSelected();**

**innerInsulation = innerInsulationCheckBox.isSelected();**

**if (zeroColourRadioButton.isSelected()) {**

**colourInput = 0;**

**}**

**if (oneColourRadioButton.isSelected()) {**

**colourInput = 1;**

**}**

**if (twoColourRadioButton.isSelected()) {**

**colourInput = 2;**

**}**

**// Creates pipe, gets the size then returns a quote.**

**this.createPipe();**

**double sizeOfPipe = this.orderedSize(lengthInMeters, diameterInInches);**

**// Prints to the text areas.**

**double costOfSelectedPipe = quote \* sizeOfPipe;**

**cartTextArea.setText(cartTextArea.getText() + "\nPipe price:  " + df.format(costOfSelectedPipe) + "\n");**

**cartTextArea.append("\nPipe characteristics:\n" + "- plastic grade: " + gradeInput + "\n- colours: " + colourInput);**

**if (chemical) {**

**cartTextArea.append("\n- chemical resistance added");**

**}**

**if (innerInsulation) {**

**cartTextArea.append("\n- inner insulation added");**

**}**

**if (outerReinforcement) {**

**cartTextArea.append("\n- outer reinforcement added");**

**}**

**cartTextArea.append("\nQuantity: " + finalQuantity + "\n");**

**double costOfSelectedOrder = costOfSelectedPipe \* finalQuantity;**

**prices.add(costOfSelectedOrder);**

**currentOrderTotal.add(costOfSelectedOrder);**

**cartTextArea.append("\nTotal: " + df.format(costOfSelectedOrder) + "\n");**

**grandTotal = this.finalTotal();**

**orderTotal = this.currentTotal();**

**currentOrderPriceLabel.setText("Â£" + df.format(orderTotal));**

**totalPriceLabel.setText("Â£" + df.format(grandTotal));**

**}**

**} catch (NumberFormatException e) {**

**if (pipeGradeComboBox.getSelectedItem() == "0") {**

**errorTextArea.append("\n" + "Please enter the pipe grade you would like");**

**} else if (buttonGroup1.getSelection() == null) {**

**errorTextArea.append("\n" + "Please select how many colours you would like");**

**} else {**

**this.wrongCharInput();**

**if (wrongChar == true) {**

**} else {**

**errorTextArea.append("Please input numeric character only!");**

**}**

**}**

**}**

**}**

**/\*\***

**\* An ActionPerformed method to check if the colour option 1 colour is**

**\* selected and then restrict or enable the user to select the extra options**

**\* they want.**

**\***

**\* @param evt used to tell the action listener an action has been performed.**

**\*/**

**private void oneColourRadioButtonActionPerformed(java.awt.event.ActionEvent evt) {**

**this.colourOptionInput();**

**}**

**/\*\***

**\* An ActionPerformed method to check if the colour option 2 colour is**

**\* selected and then restrict or enable the user to select the extra options**

**\* they want.**

**\***

**\* @param evt used to tell the action listener an action has been performed.**

**\*/**

**private void twoColourRadioButtonActionPerformed(java.awt.event.ActionEvent evt) {**

**this.colourOptionInput();**

**}**

**/\*\***

**\* A method used to reset the cart text area and write new order also also**

**\* resets current order price.**

**\***

**\* @param evt used to see if an action has occurred on the new order button**

**\* if so the code is executed.**

**\*/**

**private void newOrderButtonActionPerformed(java.awt.event.ActionEvent evt) {**

**if (currentOrderTotal.size() == 0) {**

**errorTextArea.append("\nThe cart is empty please order a pipe.");**

**} else {**

**pipeGradeComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[]{"0", "1", "2", "3", "4", "5"}));**

**this.changeStateComboBox();**

**currentOrderPriceLabel.setText("Â£0.00");**

**orderNumber++;**

**cartTextArea.append("\n--------------------------------- \n" + "Order number: " + orderNumber + "\n------------------------");**

**currentOrderTotal.clear();**

**errorTextArea.setText(null);**

**chemicalResistanceCheckBox.setEnabled(false);**

**}**

**}**

**/\*\***

**\* A method to end the order and print out to the user the final price of**

**\* the order/orders they have placed.**

**\***

**\* @param evt used to see if an action has occurred on the new order button**

**\* if so the code is executed.**

**\*/**

**private void checkoutButtonActionPerformed(java.awt.event.ActionEvent evt) {**

**pipeGradeComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[]{"0", "1", "2", "3", "4", "5"}));**

**this.changeStateComboBox();**

**currentOrderPriceLabel.setText("Â£0.00");**

**totalPriceLabel.setText("Â£0.00");**

**chemicalResistanceCheckBox.setEnabled(false);**

**if (prices.size() == 0) {**

**errorTextArea.append("\nYou have no orders! :(");**

**} else {**

**errorTextArea.setText(cartTextArea.getText());**

**errorTextArea.append("\nThank you for your order! Your total cost comes to: " + df.format(grandTotal) + "\n");**

**cartTextArea.setText(null);**

**}**

**}**

**/\*\***

**\* A method to exceptions that may break the code when the submit button is**

**\* pressed. if an error is found the correct error is posted to the user.**

**\* Checks if the size is right.**

**\*/**

**private void errors() {**

**if (lengthInMeters > 6.0 || lengthInMeters < 0.1) {**

**errorTextArea.append("\n" + "Input a length between 0.1m to 6m, please.");**

**}**

**if (diameterInInches < 1 || diameterInInches > 30) {**

**errorTextArea.append("\n" + "Please input a diameter between 1 and 30 inches");**

**}**

**if (finalQuantity > 300 || finalQuantity < 1) {**

**errorTextArea.append("\n" + "Please order at least one pipe, but less than 300");**

**}**

**error = (lengthInMeters > 6.0 || lengthInMeters < 0.10) || (diameterInInches < 1 || diameterInInches > 30) || (finalQuantity > 300 || (finalQuantity < 1));**

**}**

**/\*\***

**\* A method to exceptions that may break the code when the submit button is**

**\* pressed. if an error is found the correct error is posted to the user.**

**\* Checks if there are any wrong characters.**

**\*/**

**private void wrongCharInput() {**

**if ("".equals(lengthTextField.getText())) {**

**errorTextArea.append("\n" + "Input a length between 0.1m to 6m, please.");**

**}**

**if ("".equals(diameterTextField.getText())) {**

**errorTextArea.append("\n" + "Please input a diameter between 1 and 30 inches");**

**}**

**if ("".equals(quantityTextField.getText())) {**

**errorTextArea.append("\n" + "Please order at least one pipe, but less than 300");**

**}**

**wrongChar = (("".equals(lengthTextField.getText())) || ("".equals(diameterTextField.getText())) || ("".equals(quantityTextField.getText()))) ;**

**}**

**/\*\***

**\* A method used to determine weather a colour input has been selected then**

**\* let the user select extras based on the colour they have selected.**

**\*/**

**private void colourOptionInput() {**

**this.enableInput();**

**if (twoColourRadioButton.isSelected()) {**

**innerInsulationCheckBox.setEnabled(true);**

**} else {**

**this.disableCheckBoxes();**

**}**

**}**

**/\*\***

**\* A method used to Allow Access to the input fields by the users once one**

**\* of the radio buttons has been selected.**

**\*/**

**public void enableInput() {**

**if (buttonGroup1.getSelection() != null) {**

**lengthTextField.setEditable(true);**

**diameterTextField.setEditable(true);**

**quantityTextField.setEditable(true);**

**}**

**}**

**/\*\***

**\* A method used to disable inputs of input text fields and to reset them**

**\* back to default so the user cannot access them and the text from previous**

**\* use will be wiped.**

**\*/**

**public void disableInput() {**

**lengthTextField.setEditable(false);**

**diameterTextField.setEditable(false);**

**quantityTextField.setEditable(false);**

**lengthTextField.setText("");**

**diameterTextField.setText("");**

**quantityTextField.setText("");**

**}**

**/\*\***

**\* A method to disable JCheckBoxes if no JRadioButton is selected.**

**\*/**

**private void changeStateComboBox() {**

**errorTextArea.setText(null);**

**zeroColourRadioButton.setEnabled(false);**

**oneColourRadioButton.setEnabled(false);**

**twoColourRadioButton.setEnabled(false);**

**chemicalResistanceCheckBox.setEnabled(true);**

**chemicalResistanceCheckBox.setSelected(false);**

**buttonGroup1.clearSelection();**

**this.disableCheckBoxes();**

**this.disableInput();**

**}**

**/\*\***

**\* A method to set all JCheckBoxes to not be selected and to not be enabled.**

**\*/**

**private void disableCheckBoxes() {**

**innerInsulationCheckBox.setSelected(false);**

**innerInsulationCheckBox.setEnabled(false);**

**outerReinforcementCheckBox.setSelected(false);**

**outerReinforcementCheckBox.setEnabled(false);**

**}**

**/\*\***

**\* A method to create a Pipe based off user input.**

**\*/**

**public void createPipe() {**

**if (colourInput == 0) {**

**PipeType1 pipe = new PipeType1(gradeInput, chemical);**

**quote = pipe.getAQuote();**

**} else if (colourInput == 1) {**

**PipeType2 pipe = new PipeType2(gradeInput, chemical);**

**quote = pipe.getAQuote();**

**} else if (colourInput == 2 && innerInsulationCheckBox.isSelected() == false) {**

**PipeType3 pipe = new PipeType3(gradeInput, chemical);**

**quote = pipe.getAQuote();**

**} else if (colourInput == 2 && outerReinforcementCheckBox.isSelected() == true) {**

**PipeType5 pipe = new PipeType5(gradeInput, chemical);**

**quote = pipe.getAQuote();**

**} else {**

**PipeType4 pipe = new PipeType4(gradeInput, chemical);**

**quote = pipe.getAQuote();**

**}**

**}**

**/\*\***

**\* A method to calculate the size of a pipe to find out its cost in cubic**

**\* inches, does not calculate total cost.**

**\***

**\* @param length takes in the length in meters and converts it to inches.**

**\* @param diameter takes in the diameter in inches.**

**\* @return returns the size of the pipe.**

**\*/**

**public double orderedSize(double length, double diameter) {**

**double lengthInInches = length \* 39.37;**

**double outerRadius = diameter / 2;**

**double innerRadius = outerRadius \* 0.9;**

**double outerVolume = Math.PI \* Math.pow(outerRadius, 2) \* lengthInInches;**

**double innerVolume = Math.PI \* Math.pow(innerRadius, 2) \* lengthInInches;**

**double pipeSize = Math.pow(outerVolume - innerVolume, 2);**

**return pipeSize;**

**}**

**/\*\***

**\* A method to add the pipes in the current order the user is placing.**

**\***

**\* @return returns the current total to a Jlabel for the user to see.**

**\*/**

**public double currentTotal() {**

**double currentTotal = 0;**

**for (Double currentOrderTotal1 : currentOrderTotal) {**

**currentTotal += currentOrderTotal1;**

**}**

**return currentTotal;**

**}**

**/\*\***

**\* A method to add up the final cost of all orders placed by the user.**

**\***

**\* @return returns the final price of all orders to the user.**

**\*/**

**public double finalTotal() {**

**double finalTotal = 0;**

**for (Double price : prices) {**

**finalTotal += price;**

**}**

**return finalTotal;**

**}**

**// Variables declaration - do not modify**

**private javax.swing.ButtonGroup buttonGroup1;**

**private javax.swing.JTextArea cartTextArea;**

**private javax.swing.JButton checkoutButton;**

**private javax.swing.JCheckBox chemicalResistanceCheckBox;**

**private javax.swing.JLabel coloursLabel;**

**private javax.swing.JLabel currentOrderLabel;**

**private javax.swing.JLabel currentOrderPriceLabel;**

**private javax.swing.JLabel diameterLabel;**

**private javax.swing.JTextField diameterTextField;**

**private javax.swing.JTextArea errorTextArea;**

**private javax.swing.JLabel extrasLabel;**

**private javax.swing.JCheckBox innerInsulationCheckBox;**

**private javax.swing.JScrollPane jScrollPane1;**

**private javax.swing.JScrollPane jScrollPane2;**

**private javax.swing.JLabel lengthLabel;**

**private javax.swing.JTextField lengthTextField;**

**private javax.swing.JButton newOrderButton;**

**private javax.swing.JRadioButton oneColourRadioButton;**

**private javax.swing.JPanel orderPanel;**

**private javax.swing.JCheckBox outerReinforcementCheckBox;**

**private javax.swing.JComboBox<String> pipeGradeComboBox;**

**private javax.swing.JLabel pipeGradeLabel;**

**private javax.swing.JLabel pipeSizeLabel;**

**private javax.swing.JTextField quantityTextField;**

**private javax.swing.JLabel quntityLabel;**

**private javax.swing.JButton submitButton;**

**private javax.swing.JLabel titleLabel;**

**private javax.swing.JLabel totalLabel;**

**private javax.swing.JLabel totalPriceLabel;**

**private javax.swing.JRadioButton twoColourRadioButton;**

**private javax.swing.JRadioButton zeroColourRadioButton;**

**// End of variables declaration**

**}**

**package pipesr.us;**

**/\*\***

**\* Creates an abstract class Pipe with one abstract method getAQuote, which is**

**\* used to find the price of each type of pipe.**

**\***

**\* @author up623013**

**\* @author up674771**

**\* @version 01/12/2015**

**\*/**

**public abstract class Pipe {**

**protected double priceOfGrade;**

**protected double chemicalResistance;**

**protected double[] plasticGrade = new double[]{0.3, 0.32, 0.35, 0.4, 0.46};**

**/\*\***

**\* Default Pipe Constructor.**

**\*/**

**public Pipe() {**

**}**

**/\*\***

**\* Main Pipe Constructor, used to determine the type of pipe the user has**

**\* selected.**

**\***

**\* @param grade Takes in the grade the user has selected.**

**\* @param chemical Takes in user input for chemical resistance.**

**\*/**

**public Pipe(int grade, boolean chemical) {**

**for (int i = 1; i < 6; i++) {**

**if (grade == i) {**

**priceOfGrade = plasticGrade[i - 1];**

**}**

**chemicalResistance = chemical ? priceOfGrade \* 12 / 100 : 0.0;**

**}**

**}**

**/\*\***

**\* An abstract method to determine the cost of a pipe that the user has**

**\* asked for.**

**\***

**\* @return returns the price of the pipe.**

**\*/**

**public abstract double getAQuote();**

**}**

**package pipesr.us;**

**/\*\***

**\* A class to determine if the user inputs the correct parameters for the pipe**

**\* of type 1.**

**\***

**\* @author up623013**

**\* @author up674771**

**\* @version 01/12/2015**

**\*/**

**public class PipeType1 extends Pipe {**

**private final double gradePrice = super.priceOfGrade;**

**private final double chemicalResistance;**

**private double pipePrice;**

**/\*\***

**\* Constructs a pipe of type 1.**

**\***

**\* @param grade Takes in the user input for grade.**

**\* @param chemical Takes in user input for chemical resistance.**

**\*/**

**public PipeType1(int grade, boolean chemical) {**

**super(grade, chemical);**

**this.chemicalResistance = super.chemicalResistance;**

**}**

**/\*\***

**\* A Method to return the price of a pipe of type 1.**

**\***

**\* @return returns price of the pipe.**

**\*/**

**@Override**

**public double getAQuote() {**

**pipePrice = gradePrice + chemicalResistance;**

**return pipePrice;**

**}**

**}**

**package pipesr.us;**

**/\*\***

**\* A class to determine if the user inputs the correct parameters for the pipe**

**\* of type 2.**

**\***

**\* @author up623013**

**\* @author up674771**

**\* @version 01/12/2015**

**\*/**

**public class PipeType2 extends Pipe {**

**private final double gradePrice = super.priceOfGrade;**

**private final double chemicalResistance = super.chemicalResistance;**

**private double pipePrice;**

**private final double colour = gradePrice \* 12 / 100;**

**/\*\***

**\* Constructs a pipe of type 2.**

**\***

**\* @param grade Takes in the user input for grade.**

**\* @param chemical Takes in user input for chemical resistance.**

**\*/**

**public PipeType2(int grade, boolean chemical) {**

**super(grade, chemical);**

**}**

**/\*\***

**\* A Method to return the price of a pipe of type 2.**

**\***

**\* @return returns price of the pipe.**

**\*/**

**@Override**

**public double getAQuote() {**

**pipePrice = gradePrice + colour + chemicalResistance;**

**return pipePrice;**

**}**

**}**

**package pipesr.us;**

**/\*\***

**\* A class to determine if the user inputs the correct parameters for the pipe**

**\* of type 3.**

**\***

**\* @author up623013**

**\* @author up674771**

**\* @version 01/12/2015**

**\*/**

**public class PipeType3 extends Pipe {**

**private final double gradePrice = super.priceOfGrade;**

**private final double chemicalResistance = super.chemicalResistance;**

**private double pipePrice;**

**private final double colours = gradePrice \* 17 / 100;**

**/\*\***

**\* Constructs a pipe of type 3.**

**\***

**\* @param grade Takes in the user input for grade.**

**\* @param chemical Takes in user input for chemical resistance.**

**\*/**

**public PipeType3(int grade, boolean chemical) {**

**super(grade, chemical);**

**}**

**/\*\***

**\* A Method to return the price of a pipe of type 3.**

**\***

**\* @return returns price of the pipe.**

**\*/**

**@Override**

**public double getAQuote() {**

**pipePrice = gradePrice + colours + chemicalResistance;**

**return pipePrice;**

**}**

**}**

**package pipesr.us;**

**/\*\***

**\* A class to determine if the user inputs the correct parameters for the pipe**

**\* of type 4.**

**\***

**\* @author up623013**

**\* @author up674771**

**\* @version 01/12/2015**

**\*/**

**public class PipeType4 extends Pipe {**

**private final double gradePrice = super.priceOfGrade;**

**private final double chemicalResistance = super.chemicalResistance;**

**private double pipePrice;**

**private final double colours = gradePrice \* 17 / 100;**

**private final double innerInsulation = gradePrice \* 14 / 100;**

**/\*\***

**\* Constructs a pipe of type 4.**

**\***

**\* @param grade Takes in the user input for grade.**

**\* @param chemical Takes in user input for chemical resistance.**

**\*/**

**public PipeType4(int grade, boolean chemical) {**

**super(grade, chemical);**

**}**

**/\*\***

**\* A Method to get a pipePrice for the price of pipe type 4.**

**\***

**\* @return returns price of the pipe.**

**\*/**

**@Override**

**public double getAQuote() {**

**pipePrice = gradePrice + colours + innerInsulation + chemicalResistance;**

**return pipePrice;**

**}**

**}**

**package pipesr.us;**

**/\*\***

**\* A class to determine if the user inputs the correct parameters for the pipe**

**\* of type 5.**

**\***

**\* @author up623013**

**\* @author up674771**

**\* @version 01/12/2015**

**\*/**

**public class PipeType5 extends Pipe {**

**private final double gradePrice = super.priceOfGrade;**

**private final double chemicalResistance = super.chemicalResistance;**

**private double pipePrice;**

**private double colour = gradePrice \* 17 / 100;**

**private final double innerInsulation = gradePrice \* 14 / 100;**

**private final double outerReinforcement = gradePrice \* 15 / 100;**

**/\*\***

**\* Constructs a pipe of type 5.**

**\***

**\* @param grade Takes in the user input for grade.**

**\* @param chemical Takes in user input for chemical resistance.**

**\*/**

**public PipeType5(int grade, boolean chemical) {**

**super(grade, chemical);**

**}**

**/\*\***

**\* A Method to get a pipePrice for the price of pipe type 5.**

**\***

**\* @return returns price of the pipe.**

**\*/**

**@Override**

**public double getAQuote() {**

**pipePrice = gradePrice + colour + innerInsulation + outerReinforcement + chemicalResistance;**

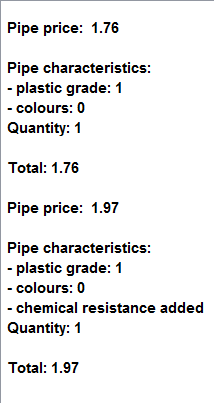
**return pipePrice;**

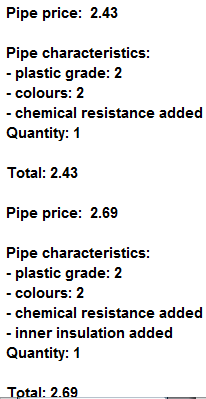
**}**

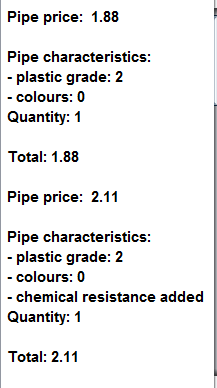
**}**

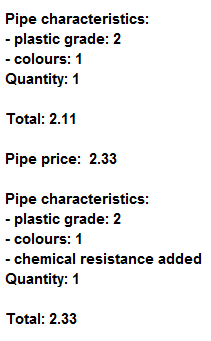
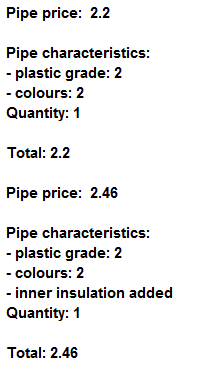
**Additional Testing Screen shots**

**All grade 1 pipes**

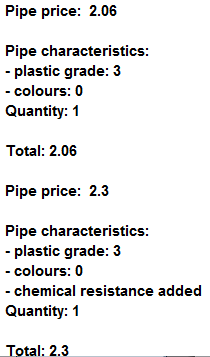
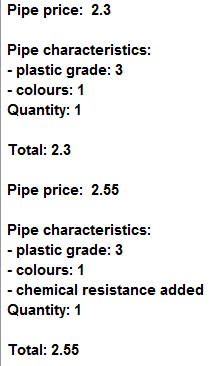
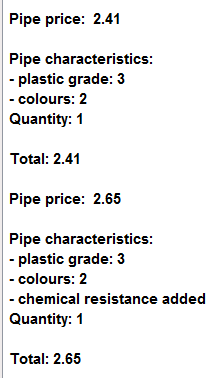
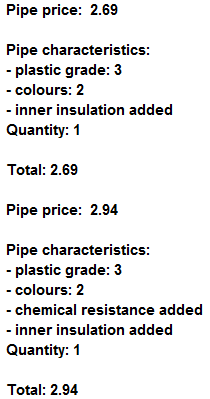
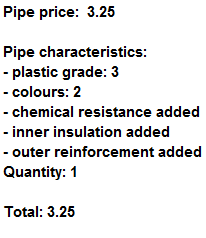


**All Grade 2 Pipes**

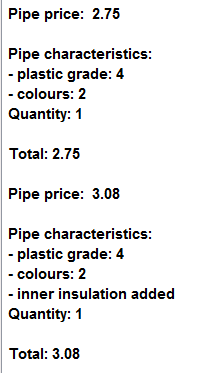
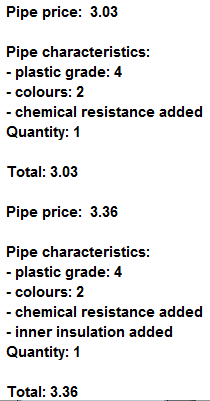
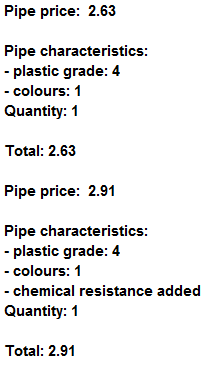
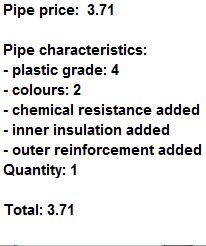




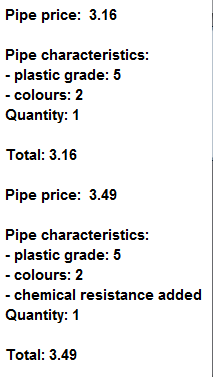
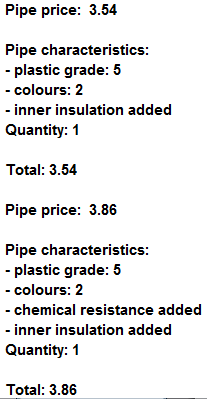
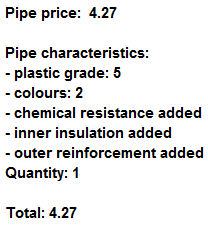
**All Grade 3 Pipes**



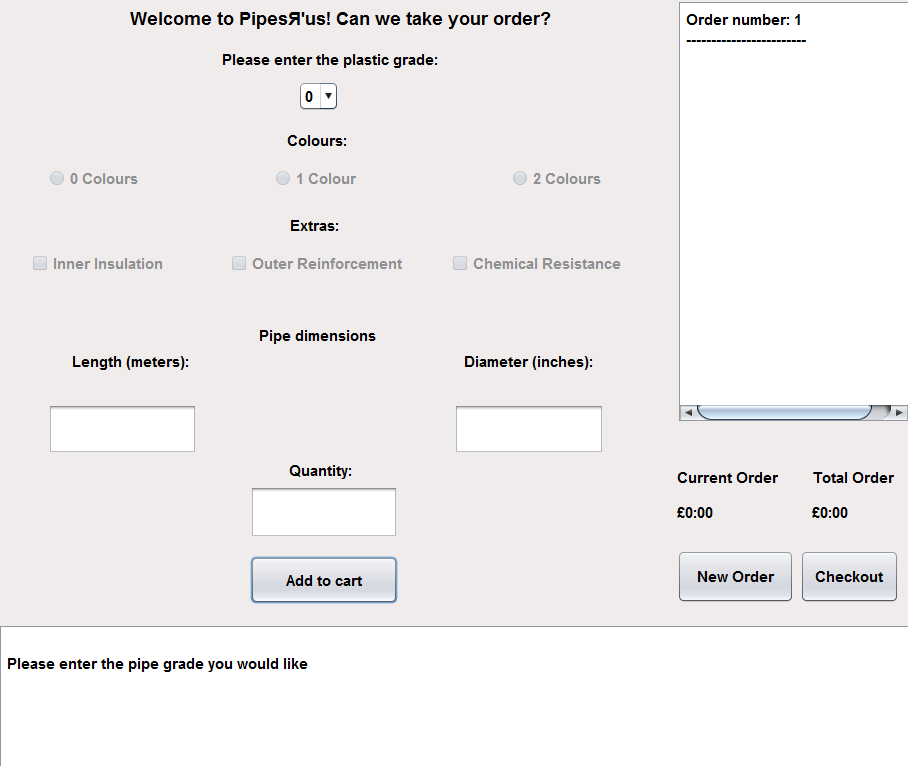
**All Grade 4 Pipes**



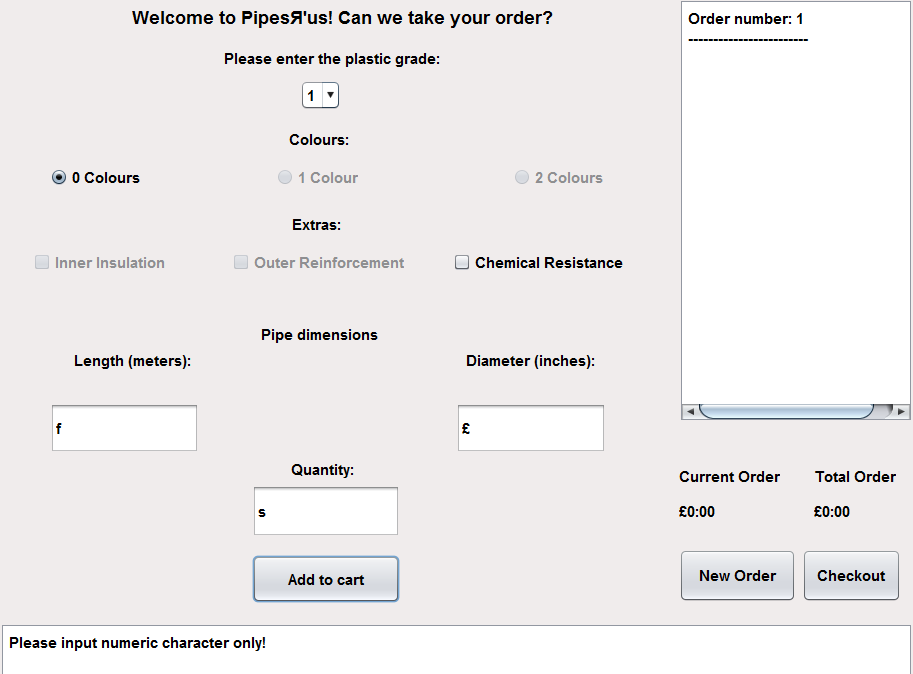
**All Grade 5 Pipes**



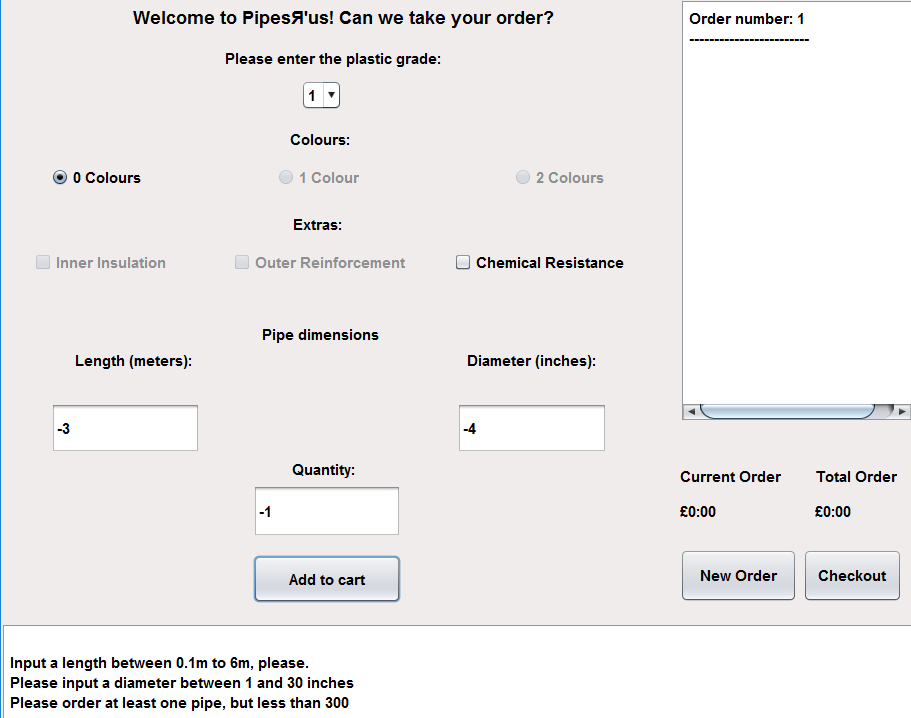
**Grade Error check**



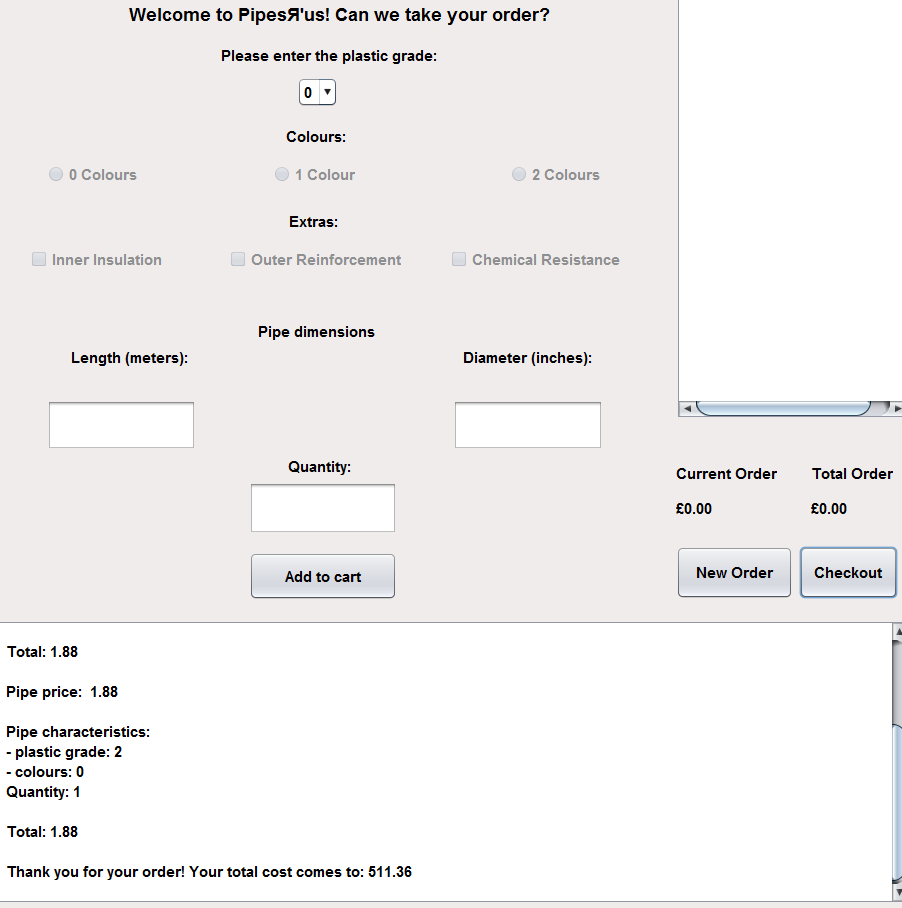
**Letters and Symbol Check**



**Minus Number Check**



**Checkout Test**



**Unfilled Measurements and Quantity Fields**

