

# ADPROC FlexBox Coursework Report

## Description

The FlexBox company produces various types of boxes that have the following properties:

- Made out of cardboard.
- Specific grades.
- Up to 2 colour printing.
- Optional reinforced bottoms.
- Optional reinforced corners.
- Optional resealable tops.

Each type of cardboard box will follow certain conditions which have to be met in order to be processed. These conditions can be seen in Table 1 in the Coursework Specification.

The base cost of the cardboard box will be dependant on the grade selected for the cardboard used to build the box as well as the total surface area measured in  $m^2$ . The pricing structure can be found in table 2 in the Coursework specification. There will also be additional costs for the cardboard boxes which will depend on how many colours the user wants printed on them, whether there are reinforced bottoms, reinforced corners and resealable tops. The pricing structure for this can be found in table 3 in the Coursework specification.

### What the system should do:

The program will ask the user multiple questions which will include:

- The dimensions of the cardboard box (Length, Width and Height).
- The grade of the cardboard.
- The amount of colours wanted (up to two colours).
- If Reinforced bottom, Reinforced corners and Resealable tops are wanted.
- How many boxes with the above conditions.
- Output the different orders of boxes.

Depending on the inputs, the system will see if they match up to the types of boxes that are available. If there are no matches, then the system will reject the inputs and display an appropriate error message and will suggest viable options based on the grade selected. If there is a match, then the Type of the box will be calculated as well as the cost of the box. Once all of the correct inputs have been entered, the total cost of the quote will be displayed to the customer. The customer should also be able to obtain more than one quote in a single session so there will be a total cost of the session. The customer should not be asked to input the Type of box wanted.

Type	Grade	Colour	Reinforced Bottom	Reinforced Corners
I	1	0	N	N
	2			
	3			
II	2	1	N	N
	3			
	4			
III	2	2	N	N
	3			
	4			
IV	2	2	Y	N
	3			
	4			
V	5	2	Y	Y
	3			
	4			

(Tab

## Assumptions

The assumptions that we have made from the requirements of the program are as follows:

- Acceptable grades, colours and reinforcements can be seen in table 1.
- The total cost is calculated by the base price of the box multiplied by the sum of the percentages of the additional features.
- The user doesn't need a table of possible valid boxes as they will be guided to correct options by error messages.
- The users are somewhat confident that they know what style of box they want and do not need to edit existing orders, such that at most only need to undo the previous order.
- The FlexBox company allows for up to and including 100 boxes per order instance and implores each order instance must have at least 1 box .
- The FlexBox company can make boxes with each dimension being between and including 0.1 - 10 metres.
- It is desired that output costs should be rounded to the nearest 2 decimal places.
- The users are confident in the cardboard grade but may get the other conditions wrong.
- A FlexBox that is created and output to the user is implicitly assumed to be valid, due to the exception handling within the program, such that the user does not need to be explicitly told that a created and output box is valid.

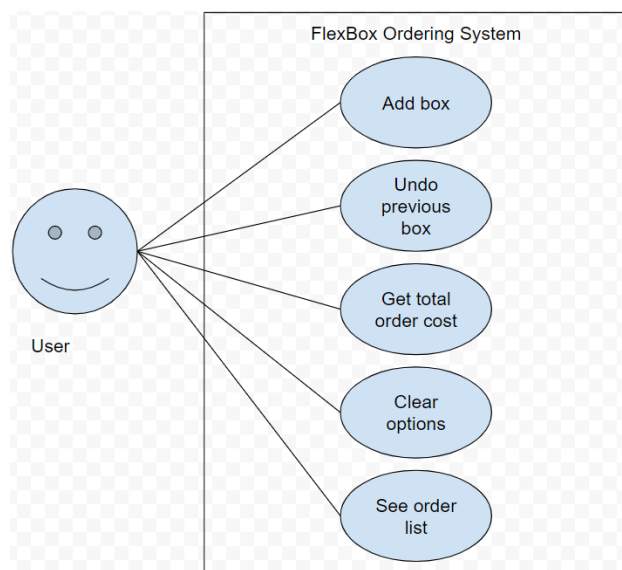
## Limitations

- The program doesn't allow for amendment of boxes after adding it to the output box.
- The program doesn't save the information to come back to after the program has terminated.
- The FlexBox company impose a limit on the types of boxes that can be created.

## UML Modelling

### Use Case Diagram

The Use Case Diagram can be seen in Figure 1. This shows that the system will have five main features which are Add Box, Remove Box, Get Total Order Cost, Clear All Boxes and See Order List. The Add Box and the Remove Box features will be explained further in the Class Hierarchy. The Get Total Order Cost feature will have two sub features that will get the cost of the order of a certain box and the cost of the session that contains multiple orders. The See Order List



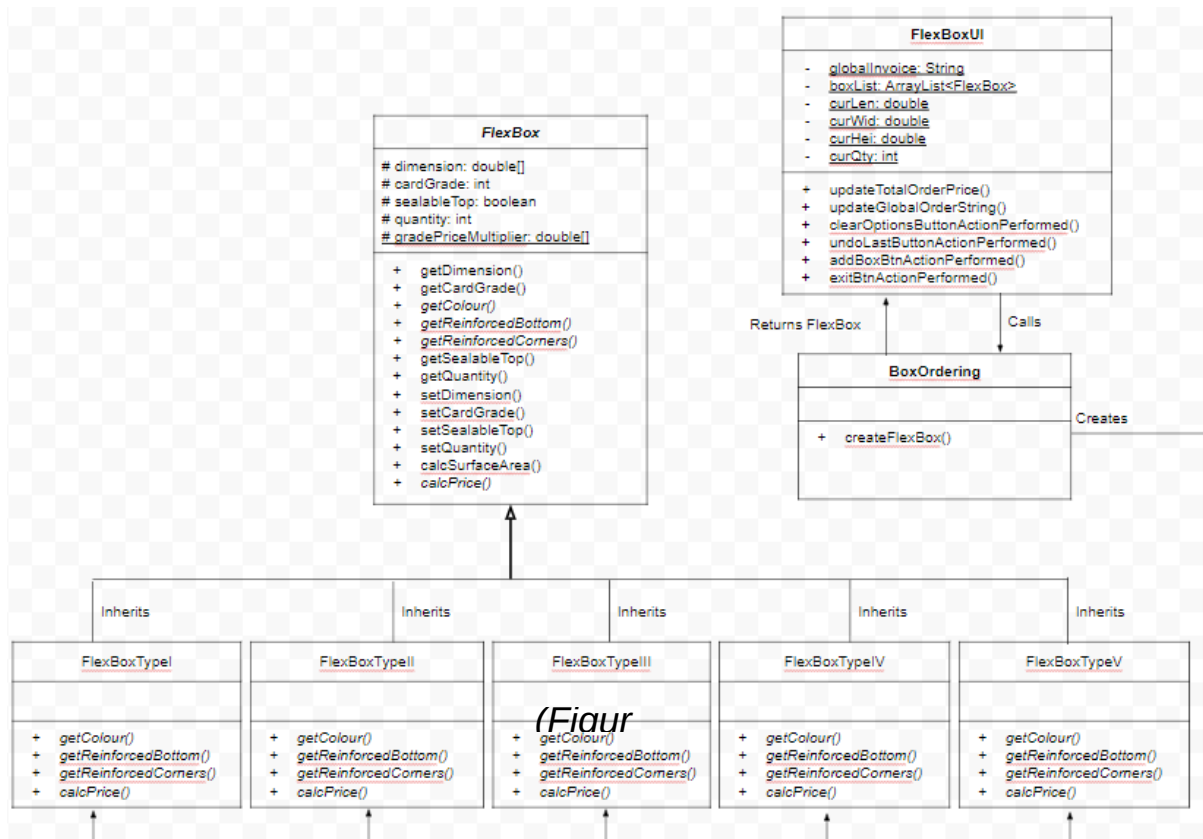
(Figure

feature will allow the customer to see what their orders and session looks like to check that they have the desired outputs.

## Class Hierarchy Diagram

Figure 2 shows the class hierarchy. This models the classes that we have created for the system, describing their attributes and methods. There will be one superclass called 'FlexBox' that will contain almost all of the properties and methods that we will need and the subclasses will only have one method that will calculate the cost of the order, and methods to return class dependant values (colour printing, reinforced bottom/corner).

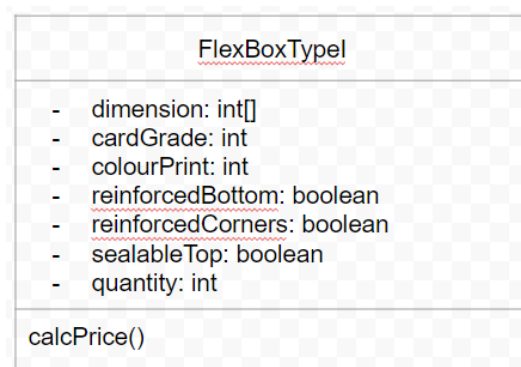
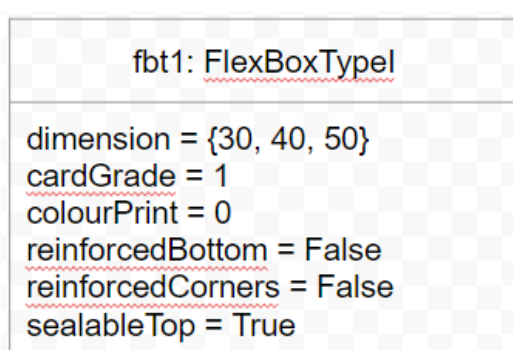
The FlexBox class is declared as abstract and declares four abstract methods, indicated by italics in the class hierarchy diagram. Each subclass implements the four abstract methods, to suit the needs of the subclass.



## Class/Instance Diagram

Figure 3 illustrates the class for FlexBoxTypeI. It is instantiated if the FlexBox options entered satisfy the conditions needed for a Type I FlexBox.

Figure 4 demonstrates an instance of a FlexBox of Type I with the possible field values that an instance of FlexBoxTypeI can have.



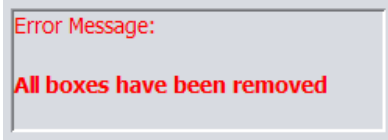
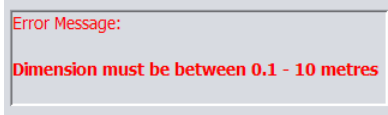
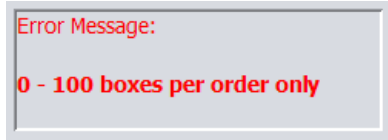
# Sample Input / Output Screenshots

## Considering valid inputs

Boxes	Inputs	Output	Total Cost
1	<div> <div> Dimensions (m):  Length <input type="text" value="1"/>  Width <input type="text" value="2"/>  Height <input type="text" value="3"/> </div> <div> Additional Features:  <input type="checkbox"/> Reinforced Corners  <input type="checkbox"/> Reinforced Bottom  <input checked="" type="checkbox"/> Sealable Top </div> <div> Cardboard Grade:  <input checked="" type="radio"/> Grade 1  <input type="radio"/> Grade 2  <input type="radio"/> Grade 3  <input type="radio"/> Grade 4  <input type="radio"/> Grade 5 </div> <div> Number of Colours: <input type="text" value="0"/> </div> <div> Quantity of Boxes in this style:  <input type="text" value="1"/> </div> </div>	<div> Box Order Number: 1  Box Dimension: [1.0, 2.0, 3.0]  Card Grade: 1  Colours Printed: 0  Reinforced Bottoms?: false  Reinforced Corners?: false  Sealable Top?: true  Quantity of this box style ordered: 1  ORDER SUB-TOTAL: £13.31 </div>	<div> <b>Total Order Price: £13.31</b> </div>
2	<div> <div> Dimensions (m):  Length <input type="text" value="3"/>  Width <input type="text" value="2"/>  Height <input type="text" value="1"/> </div> <div> Additional Features:  <input checked="" type="checkbox"/> Reinforced Corners  <input checked="" type="checkbox"/> Reinforced Bottom  <input type="checkbox"/> Sealable Top </div> <div> Cardboard Grade:  <input type="radio"/> Grade 1  <input type="radio"/> Grade 2  <input type="radio"/> Grade 3  <input type="radio"/> Grade 4  <input checked="" type="radio"/> Grade 5 </div> <div> Number of Colours: <input type="text" value="2"/> </div> <div> Quantity of Boxes in this style:  <input type="text" value="1"/> </div> </div>	<div> Box Order Number: 1  Box Dimension: [1.0, 2.0, 3.0]  Card Grade: 1  Colours Printed: 0  Reinforced Bottoms?: false  Reinforced Corners?: false  Sealable Top?: true  Quantity of this box style ordered: 1  ORDER SUB-TOTAL: £13.31 </div> <div> Box Order Number: 2  Box Dimension: [3.0, 2.0, 1.0]  Card Grade: 5  Colours Printed: 2  Reinforced Bottoms?: true  Reinforced Corners?: true  Sealable Top?: false  Quantity of this box style ordered: 1  ORDER SUB-TOTAL: £46.20 </div>	<div> <b>Total Order Price: £59.51</b> </div>

## Considering Invalid Inputs

When non-numerals are entered into the dimensions fields:	<div> Error Message:  <b>Dimension values must be numerals</b> </div>
When a card grade isn't selected:	<div> Error Message:  <b>Card Grade must be selected</b> </div>
When the quantity isn't an integer:	<div> Error Message:  <b>Quantity value must be an integer</b> </div>
When a box cannot be made and card grade is 1. The additional message is	<div> Error Message:  <b>We cannot make this box</b>  <b>Boxes with grade 1 card must have 0 colour and no reinforced bottom or corners.</b> </div>

specialised for each card grade that is selected to assist the user in creating a valid box:	
When user tries to undo boxes when all have been undone:	
If entered dimension is outside the range of 0.1 - 10:	
If entered quantity is outside the range of 1-100:	

## Test Schedule

The test plan aims to demonstrate the robustness of our program by evaluating seven different types of use cases that consider the possible ways in which a user can use the system. These are: correct inputs (Test 1, Test 2), inputs that cannot create a valid FlexBox but are otherwise correct (Test 3), inputs that create a number format exception (Test 4, Test 5), and inputs that are out of an assumed range (Test 6, Test 7).

There will be one test for the out-of-range inputs where, in the FlexBox options section, we will input values that cannot create a valid box. We also did two tests where, in the options that take typed user input (dimension and quantity), we enter in a character that should not be processed to check the robustness of taking in user inputs; to see if the system rejects that input and ask for a valid input. A further test we conducted evaluated the effectiveness of handling user inputs that are outside of the assumed range of acceptable values for the dimension and quantity values.

By using try/catch blocks we are able to throw distinct exceptions which enable the program to handle the exceptions in specific ways to help guide the user in effective FlexBox creation.

The output screenshots can be seen in the appendix.

---

## Appendix

### Group Contribution Form

Group No GrC-8:

Student number and contribution:

1. UP850844      25/100
2. UP787839      25/100
3. UP767896      25/100
4. UP879045      25/100

## Test Schedule Screenshots

Test 1 Outputs:

Box Order Number: 1  
Box Dimension: [2.0, 3.0, 4.0]  
Card Grade: 1  
Colours Printed: 0  
Reinforced Bottoms?: false  
Reinforced Corners?: false  
Sealable Top?: true  
Quantity of this box style ordered: 1  
ORDER SUB-TOTAL: £31.46

**Total Order Price: £31.46**

Test 2 Outputs:

Box Order Number: 1  
Box Dimension: [1.5, 2.5, 3.5]  
Card Grade: 4  
Colours Printed: 2  
Reinforced Bottoms?: true  
Reinforced Corners?: true  
Sealable Top?: false  
Quantity of this box style ordered: 1  
ORDER SUB-TOTAL: £48.71

**Total Order Price: £48.71**

## Test 3 Outputs:

The screenshot shows the FlexBox Ordering System interface. The dimensions are set to Length: 2, Width: 3, and Height: 4. Under Additional Features, Reinforced Corners and Reinforced Bottom are unchecked, while Sealable Top is checked. Cardboard Grade is set to Grade 4 (selected). The Number of Colours is 0, and the Quantity of Boxes in this style is 1. The Total Order Price is £0.00. An error message is displayed: "We cannot make this box. Boxes with grade 4 card must have 2 colours, a reinforced bottom, but with no reinforced corners." The interface includes buttons for Add Box, Clear Options, Undo Last Box, and Exit.

FlexBox Ordering System

Dimensions (m):  
Length: 2  
Width: 3  
Height: 4

Additional Features:  
☐ Reinforced Corners  
☐ Reinforced Bottom  
☒ Sealable Top

Cardboard Grade:  
☐ Grade 1  
☐ Grade 2  
☐ Grade 3  
☒ Grade 4  
☐ Grade 5

Number of Colours: 0

Quantity of Boxes in this style: 1

Total Order Price: £0.00

Error Message:  
We cannot make this box  
Boxes with grade 4 card must have 2 colours, a reinforced bottom, but with no reinforced corners.

Exit

## Test 4 Outputs:

The screenshot shows the FlexBox Ordering System interface. The dimensions are set to Length: length, Width: 3, and Height: 4. Under Additional Features, Reinforced Corners and Reinforced Bottom are unchecked, while Sealable Top is checked. Cardboard Grade is set to Grade 1 (selected). The Number of Colours is 0, and the Quantity of Boxes in this style is 1. The Total Order Price is £0.00. An error message is displayed: "Dimension values must be numerals". The interface includes buttons for Add Box, Clear Options, Undo Last Box, and Exit.

FlexBox Ordering System

Dimensions (m):  
Length: length  
Width: 3  
Height: 4

Additional Features:  
☐ Reinforced Corners  
☐ Reinforced Bottom  
☒ Sealable Top

Cardboard Grade:  
☒ Grade 1  
☐ Grade 2  
☐ Grade 3  
☐ Grade 4  
☐ Grade 5

Number of Colours: 0

Quantity of Boxes in this style: 1

Total Order Price: £0.00

Error Message:  
Dimension values must be numerals

Exit

## Test 5 Outputs:

The screenshot shows the FlexBox Ordering System window. The interface includes several input fields and checkboxes. The 'Dimensions (m)' section has Length: 6, Width: 3, and Height: 4. The 'Additional Features' section has checkboxes for 'Reinforced Corners', 'Reinforced Bottom', and 'Sealable Top' (checked). The 'Cardboard Grade' section has radio buttons for Grade 1 (selected), Grade 2, Grade 3, Grade 4, and Grade 5. The 'Number of Colours' is set to 0. The 'Quantity of Boxes in this style' is set to 0. The 'Total Order Price' is £0.00. The 'Error Message' section displays the message: 'Quantity value must be an integer'. The 'Exit' button is visible at the bottom.

FlexBox Ordering System

Dimensions (m):  
Length: 6  
Width: 3  
Height: 4

Additional Features:  
☐ Reinforced Corners  
☐ Reinforced Bottom  
☒ Sealable Top

Cardboard Grade:  
☒ Grade 1  
☐ Grade 2  
☐ Grade 3  
☐ Grade 4  
☐ Grade 5

Number of Colours: 0

Quantity of Boxes in this style:  
0

Add Box  
Clear Options  
Undo Last Box

Total Order Price: £0.00

Error Message:  
Quantity value must be an integer

Exit

## Test 6 Outputs:

The screenshot shows the FlexBox Ordering System window. The interface includes several input fields and checkboxes. The 'Dimensions (m)' section has Length: 6, Width: 3, and Height: 4. The 'Additional Features' section has checkboxes for 'Reinforced Corners', 'Reinforced Bottom', and 'Sealable Top' (checked). The 'Cardboard Grade' section has radio buttons for Grade 1 (selected), Grade 2, Grade 3, Grade 4, and Grade 5. The 'Number of Colours' is set to 0. The 'Quantity of Boxes in this style' is set to 1000. The 'Total Order Price' is £0.00. The 'Error Message' section displays the message: '1 - 100 boxes per order only'. The 'Exit' button is visible at the bottom.

Dimensions (m):  
Length: 6  
Width: 3  
Height: 4

Additional Features:  
☐ Reinforced Corners  
☐ Reinforced Bottom  
☒ Sealable Top

Cardboard Grade:  
☒ Grade 1  
☐ Grade 2  
☐ Grade 3  
☐ Grade 4  
☐ Grade 5

Number of Colours: 0

Quantity of Boxes in this style:  
1000

Add Box  
Clear Options  
Undo Last Box

Total Order Price: £0.00

Error Message:  
1 - 100 boxes per order only



## Test 7 Outputs:

## Authored Source Code

### From FlexBoxUI Class

#### GUI Class Main Method

```
public static void main(String args[]) {
//Auto-generated code here
java.awt.EventQueue.invokeLater(new Runnable() {
    public void run() {
        new FlexBoxUI().setVisible(true);
    }
});
}
```

#### Add Box Button Action Performed Method

```
/**
 * Method to add a FlexBox to the total order based on the
selected options within the GUI
 * @param evt
 */
@SuppressWarnings("empty-statement")
private void addBoxBtnActionPerformed(java.awt.event.ActionEvent
evt) {
    errorLabel.setText("");
    changeToSuitGrade.setText("");
    try{//handles valid creation of a flexbox
        try{//handles validation of dimension values
            curLen = Double.valueOf(length.getText());
            curWid = Double.valueOf(width.getText());
```

```

        curHei = Double.valueOf(height.getText());
        if(
            (curLen > 10 || curLen < 0.1) ||
            (curWid > 10 || curWid < 0.1) ||
            (curHei > 10 || curHei < 0.1)
        ){
            errorLabel.setText("Dimension must be between
0.1 - 10 metres");
            throw new Exception();
        }
    }catch(Exception e){
        if(e instanceof NumberFormatException){
            errorLabel.setText("Dimension values must be
numerals");
        }
        throw new Exception();
    }
    double[] dim={curLen, curWid, curHei};
    int grade;
    if(grade1.isSelected()){
        grade = 1;
    }else if (grade2.isSelected()){
        grade = 2;
    }else if (grade3.isSelected()){
        grade = 3;
    }else if (grade4.isSelected()){
        grade = 4;
    }else if (grade5.isSelected()){
        grade = 5;
    }else{
        errorLabel.setText("Card Grade must be selected");
        throw new Exception();
    }
    int colours =
Integer.valueOf(coloursDrop.getSelectedItem().toString());
    boolean rBot = rBotCheck.isSelected();
    boolean rCorn = rCornCheck.isSelected();
    boolean sTop = sTopCheck.isSelected();
    try{//handles validation of the quantity value
        curQty = Integer.valueOf(quantity.getText());
        if(curQty < 1 || curQty > 100){
            errorLabel.setText("1 - 100 boxes per order
only");
            throw new Exception();
        }
    }catch(Exception e){
        if(e instanceof NumberFormatException){

```

```

        errorLabel.setText("Quantity value must be an
integer");
    }
    throw new Exception();
}
FlexBox newFB = BoxOrdering.createFlexBoxOrder(dim,
grade, colours, rBot, rCorn, sTop, curQty);
if(newFB != null){
    boxList.add(newFB);
}else{
    changeToSuitGrade.setText("");
    errorLabel.setText("We cannot make this box");
    switch(grade){//handles the error message presented
to assist the user to create a valid flexbox
        case 1: changeToSuitGrade.setText("Boxes with
grade 1 card must "
                                           + "have 0 colour
and no reinforced "
                                           + "bottom or
corners.");
            break;
        case 2: changeToSuitGrade.setText("Boxes with
grade 2 card must "
                                           + "have 1 colour
and no reinforced "
                                           + "bottom or
corners.");
            break;
        case 3: changeToSuitGrade.setText("Boxes with
grade 3 card must "
                                           + "have 2
colours and no reinforced "
                                           + "bottom or
corners.");
            break;
        case 4: changeToSuitGrade.setText("Boxes with
grade 4 card must "
                                           + "have 2
colours, a reinforced "
                                           + "bottom, but
with no reinforced corners.");
            break;
        case 5: changeToSuitGrade.setText("Boxes with
grade 5 card must "
                                           + "have 2
colours, a reinforced "
                                           + "bottom and
reinforced corners.");
    }
}

```

```

                break;
            }
            throw new Exception();
        }
        // operations past here are only complete if a box has
        been successfully created
        errorLabel.setText("");
        changeToSuitGrade.setText("");
        updateGlobalOrderString();
        updateTotalOrderPrice();
    }catch (Exception e){}
}

```

### Undo Last Box Button Action Performed Method

```

/**
 * Method to undo the creation of the previously created FlexBox
 * @param evt
 */
private void
undoLastButtonActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        boxList.remove(boxList.size() - 1);
        updateGlobalOrderString();
        updateTotalOrderPrice();
    }catch(Exception e){
        if(e instanceof
java.lang.ArrayIndexOutOfBoundsException){
            errorLabel.setText("All boxes have been undone");
        }
    }
}

```

### Clear Options Button Action Performed Method

```

/**
 * Method to clear the selected options within the GUI
 * @param evt
 */
private void
clearOptionsButtonActionPerformed(java.awt.event.ActionEvent evt) {
    errorLabel.setText("");
    changeToSuitGrade.setText("");
    length.setText("");
    width.setText("");
    height.setText("");
    gradeButtonGroup.clearSelection();
    coloursDrop.setSelectedIndex(0);
}

```

```

        rBotCheck.setSelected(false);
        rCornCheck.setSelected(false);
        sTopCheck.setSelected(false);
        quantity.setText("");
    }

```

#### Update Total Order Price Method

```

/**
 *Method to update the total order price
 */
public void updateTotalOrderPrice(){
    double newPrice = 0;
    for (int i = 0; i < boxList.size(); i++) {
        newPrice += boxList.get(i).calcPrice();
    }
    totalLabel.setText("Total Order Price: £" +
String.format("%.2f", newPrice));
}

```

#### Update Global Order String Method

```

/**
 * Method to update the order string used as output for the
customer
 */
public void updateGlobalOrderString(){
    globalInvoice = "";
    for (int i = 0; i < boxList.size(); i++) {
        globalInvoice += "Box Order Number: " + (i+1);
        String boxStatement = boxList.get(i).orderStatement();
        globalInvoice += "\n" + boxStatement;
    }
    outputTextArea.setText(globalInvoice);
}

```

#### Exit Button Action Performed

```

/**
 * Button to exit the program
 * @param evt
 */
private void exitBtnActionPerformed(java.awt.event.ActionEvent
evt) {
    System.exit(0);
}

```

#### GUI components declaration

```

private javax.swing.JButton addBoxBtn;

```

```

private javax.swing.JLabel addFeatLabel;
private javax.swing.JPanel addFeatPanel;
private javax.swing.JPanel boxButtonPanel;
private javax.swing.JLabel changeToSuitGrade;
private javax.swing.JButton clearOptionsButton;
private javax.swing.JComboBox<String> coloursDrop;
private javax.swing.JPanel coloursPanel;
private javax.swing.JLabel dimLabel;
private javax.swing.JPanel dimPanel;
private javax.swing.JLabel errMessLabel;
private javax.swing.JLabel errorLabel;
private javax.swing.JPanel errorPanel;
private javax.swing.JButton exitBtn;
private javax.swing.JRadioButton grade1;
private javax.swing.JRadioButton grade2;
private javax.swing.JRadioButton grade3;
private javax.swing.JRadioButton grade4;
private javax.swing.JRadioButton grade5;
private javax.swing.ButtonGroup gradeButtonGroup;
private javax.swing.JLabel gradeLabel;
private javax.swing.JPanel gradePanel;
private javax.swing.JLabel heiLabel;
private javax.swing.JTextField height;
private javax.swing.JScrollPane jScrollPane1;
private javax.swing.JLabel lenLabel;
private javax.swing.JTextField length;
private javax.swing.JPanel mainPanel;
private javax.swing.JLabel numColLabel;
private javax.swing.JPanel outputPanel;
private javax.swing.JTextArea outputTextArea;
private javax.swing.JLabel qtyLabel;
private javax.swing.JPanel qtyPanel;
private javax.swing.JTextField quantity;
private javax.swing.JCheckBox rBotCheck;
private javax.swing.JCheckBox rCornCheck;
private javax.swing.JCheckBox sTopCheck;
private javax.swing.JLabel totalLabel;
private javax.swing.JPanel totalPricePanel;
private javax.swing.JButton undoLastButton;
private javax.swing.JLabel widLabel;
private javax.swing.JTextField width;

```

### **GUI class imports, instance variables and constructor**

```

package adproc;
import java.awt.*;
import java.util.ArrayList;
/**

```

```

    * FlexBoxUI class to manage the design and fuctionality of the GUI
    */
public class FlexBoxUI extends javax.swing.JFrame {
    private String globalInvoice = "";
    private ArrayList<FlexBox> boxList = new ArrayList<FlexBox>();
    private double curLen;
    private double curWid;
    private double curHei;
    private int curQty;
    /**
     * Creates new form FlexBoxUI
     */
    public FlexBoxUI() {
        initComponents();
        setTitle("FlexBox Ordering System");
        setLayout(new FlowLayout());
        outputTextArea.setEditable(false);

    }
    ...

```

### FlexBox superclass

```

package adproc;
import java.util.Arrays;
/**
 * FlexBox superclass
 *
 */
public abstract class FlexBox {
    protected double[] dim;
    protected int cGrade;
    protected boolean sTop;
    protected int qty;
    protected static double[] gradePriceMultiplier = {0.55, 0.65,
0.82, 0.98,
                                1.5};

    //values for colours, reinforced bottom and reinforced corners
    are accessed by calling methods implemented in each subclass

    /**
     * Default constructor for FlexBox
     */
    public FlexBox(){}
    /**
     * Constructor for FlexBox using all variables
     * @param dimension Array for the x,y,z dimensions of the box
     * @param cardGrade Value of the card grade (1-5)
     * @param sealableTop Declares whether box has a sealable top

```

```

    * @param quantity Quantity of boxes of this instance
    */
    public FlexBox(double[] dimension, int cardGrade,
                  boolean sealableTop, int quantity){
        dim = dimension;
        cGrade = cardGrade;
        sTop = sealableTop;
        qty = quantity;
    }
    /**
     * Dimension array accessor method
     * @return box dimension array
     */
    public double[] getDimension(){
        return dim;
    }
    /**
     * Card grade accessor method
     * @return box card grade
     */
    public int getCardGrade(){
        return cGrade;
    }
    /**
     * Accessor method to get number of colours printed on box
     * @return number of colours printed on box
     */
    abstract int getColour();
    /**
     * Accessor method to get reinforced bottom value
     * @return true if box has reinforced bottom
     */
    abstract boolean getReinforcedBottom();
    /**
     * Accessor method to get reinforced corners value
     * @return true if box has reinforced corners
     */
    abstract boolean getReinforcedCorners();
    /**
     * Accessor method to get sealable top value
     * @return true if box has sealable top
     */
    public boolean getSealableTop(){
        return sTop;
    }
    /**
     * Accessor method to get box order quantity value
     * @return quantity of boxes ordered of the specific instance

```



```

    */
    public int getQuantity(){
        return qty;
    }
    /**
     * Mutator method to set dimension array
     * @param newDim the array for dimension
     */
    public void setDimension(double[] newDim){
        dim = newDim;
    }
    /**
     * Mutator method to set card grade value
     * @param newGrade card grade value
     */
    public void setCardGrade(int newGrade){
        cGrade = newGrade;
    }
    /**
     * Mutator method to set whether box has a sealable top
     * @param newSTop false if box doesn't have a sealable top
     */
    public void setSealableTop(boolean newSTop){
        sTop = newSTop;
    }
    /**
     * Mutator method to set box order quantity value
     * @param newQuantity new value for the quantity of boxes of
this instance
     */
    public void setQuantity(int newQuantity){
        qty = newQuantity;
    }
    /**
     * Method to calculate the surface area of the box
     * @return surface area value of box
     */
    public double calcSurfaceArea(){
        double sa;
        sa = (2 * dim[0] * dim[2]) +
            (2 * dim[1] * dim[0]) +
            (2 * dim[2] * dim[1]);
        return sa;
    }
    /**
     * Abstract method to calculate the base price (card determined)
of the box
     * @return base price value of the box

```

```

    */
    abstract double calcPrice();
    /**
     * Method to create and return a string describing the features
of an ordered box
     * @return statement about an flexbox order
     */
    public String orderStatement(){
        String orderSta = "Box Dimension: " + Arrays.toString(dim);
        orderSta += "\nCard Grade: " + cGrade;
        orderSta += "\nColours Printed: " + getColour();
        orderSta += "\nReinforced Bottoms?: " +
getReinforcedBottom();
        orderSta += "\nReinforced Corners?: " +
getReinforcedCorners();
        orderSta += "\nSealable Top?: " + sTop;
        orderSta += "\nQuantity of this box style ordered: " + qty;
        orderSta += "\nORDER SUB-TOTAL: £" + String.format("%.2f",
calcPrice()) + "\n\n";
        return orderSta;
    }
}

```

### FlexBoxTypeI subclass

```

package adproc;
/**
 * FlexBoxTypeI subclass
 * @author up850844
 */
public class FlexBoxTypeI extends FlexBox{
    /**
     * Constructor for FlexBoxTypeI subclass
     * @param dimension Array for the x,y,z dimensions of the box
     * @param cardGrade Value of the card grade (1-5)
     * @param sealableTop Declares whether box has a sealable top
     * @param quantity Quantity of boxes of this instance
     */
    public FlexBoxTypeI(double[] dimension, int cardGrade,
        boolean sealableTop, int quantity){
        super(dimension, cardGrade, sealableTop, quantity);
    }
    /**
     * Method to return value of colour for TypeI FlexBox
     * @return Will be 0 for all TypeI FlexBoxes
     */
    @Override
    public int getColour(){

```

```

        return 0;
    }
    /**
     * Method to return value of reinforced bottom for TypeI FlexBox
     * @return Will always be false for TypeI FlexBoxes
     */
    @Override
    public boolean getReinforcedBottom(){
        return false;
    }
    /**
     * Method to return value of reinforced corners for TypeI
FlexBox
     * @return Will always be false for TypeI FlexBoxes
     */
    @Override
    public boolean getReinforcedCorners(){
        return false;
    }
    /**
     * Method to calculate the order price for the box(es) of this
instance
     * @return order price for the box(es) of this instance
     */
    @Override
    public double calcPrice(){
        double area = calcSurfaceArea();
        double p = area * gradePriceMultiplier[cGrade-1];
        if(sTop == true){
            return p * 1.1 * qty;
        }else{
            return p * qty;
        }
    }
}

```

### FlexBoxTypeII subclass

```

package adproc;
/**
 * FlexBoxTypeII subclass
 */
public class FlexBoxTypeII extends FlexBox {
    /**
     * Constructor for the FlexBoxTypeII subclass
     * @param dimension Array for the x,y,z dimensions of the box
     * @param cardGrade Value of the card grade (1-5)
     * @param sealableTop Declares whether box has a sealable top

```

```

    * @param quantity Quantity of boxes of this instance
    */
    public FlexBoxTypeII(double[] dimension, int cardGrade,
        boolean sealableTop, int quantity){
        super(dimension, cardGrade, sealableTop, quantity);
    }
    /**
    * Method to return value of colour for TypeII FlexBox
    * @return Will be 1 for all TypeII FlexBoxes
    */
    @Override
    public int getColour(){
        return 1;
    }
    /**
    * Method to return value of reinforced bottom for TypeII
FlexBox
    * @return Will be false for all TypeII FlexBoxes
    */
    @Override
    public boolean getReinforcedBottom(){
        return false;
    }
    /**
    * Method to return value of reinforced corners for TypeII
FlexBox
    * @return Will be false for all TypeII FlexBoxes
    */
    @Override
    public boolean getReinforcedCorners(){
        return false;
    }
    /**
    * Method to calculate the order price for the box(es) of this
instance
    * @return order price for the box(es) of this instance
    */
    @Override
    public double calcPrice(){
        double area = calcSurfaceArea();
        double p = area * gradePriceMultiplier[cGrade-1];

        p *= (1 + 0.12);
        if(sTop == true){
            return p * 1.1 * qty;
        }else{
            return p * qty;
        }
    }

```

```

    }
}

```

### FlexBoxTypeIII subclass

```

package adproc;
/**
 * FlexBoxTypeIII subclass
 */
public class FlexBoxTypeIII extends FlexBox {
    /**
     * Constructor for FlexBoxTypeIII subclass
     * @param dimension Array for the x,y,z dimensions of the box
     * @param cardGrade Value of the card grade (1-5)
     * @param sealableTop Declares whether box has a sealable top
     * @param quantity Quantity of boxes of this instance
     */
    public FlexBoxTypeIII(double[] dimension, int cardGrade,
                          boolean sealableTop, int quantity){
        super(dimension, cardGrade, sealableTop, quantity);
    }
    /**
     * Method to return value of colour for TypeIII FlexBox
     * @return Will be 2 for all TypeIII FlexBoxes
     */
    @Override
    public int getColour(){
        return 2;
    }
    /**
     * Method to return value of reinforced bottom for TypeIII
FlexBox
     * @return Will be false for all TypeIII FlexBoxes
     */
    @Override
    public boolean getReinforcedBottom(){
        return false;
    }
    /**
     * Method to return value of reinforced corners for TypeIII
FlexBox
     * @return Will be false for all TypeIII FlexBoxes
     */
    @Override
    public boolean getReinforcedCorners(){
        return false;
    }
}
/**

```

```

        * Method to calculate the order price for the box(es) of this
instance
        * @return order price for the box(es) of this instance
        */
@Override
public double calcPrice(){
    double area = calcSurfaceArea();
    double p = area * gradePriceMultiplier[cGrade-1];

    p *= (1 + 0.15);
    if(sTop == true){
        return p * 1.1 * qty;
    }else{
        return p * qty;
    }
}
}

```

### FlexBoxTypeIV subclass

```

package adproc;
/**
 * FlexBoxTypeIV subclass
 */
public class FlexBoxTypeIV extends FlexBox {
    /**
     * Constructor for FlexBoxTypeIV subclass
     * @param dimension Array for the x,y,z dimensions of the box
     * @param cardGrade Value of the card grade (1-5)
     * @param sealableTop Declares whether box has a sealable top
     * @param quantity Quantity of boxes of this instance
     */
    public FlexBoxTypeIV(double[] dimension, int cardGrade,
                        boolean sealableTop, int quantity){
        super(dimension, cardGrade, sealableTop, quantity);
    }
    /**
     * Method to return value of colour for TypeIV FlexBox
     * @return Will be 2 for all TypeIV FlexBoxes
     */
    @Override
    public int getColour(){
        return 2;
    }
    /**
     * Method to return value of reinforced bottom for TypeIV
FlexBox
     * @return Will be true for all TypeIV FlexBoxes

```

```

        */
        @Override
        public boolean getReinforcedBottom(){
            return true;
        }
        /**
         * Method to return value of reinforced corners for TypeIV
FlexBox
         * @return Will be false for all TypeIV FlexBoxes
         */
        @Override
        public boolean getReinforcedCorners(){
            return false;
        }

        /**
         * Method to calculate the order price for the box(es) of this
instance
         * @return order price for the box(es) of this instance
         */
        @Override
        public double calcPrice(){
            double area = calcSurfaceArea();
            double p = area * gradePriceMultiplier[cGrade-1];

            double totalModifier = 1 + 0.15 + 0.13;
            p *= totalModifier;
            if(sTop == true){
                return p * 1.1 * qty;
            }else{
                return p * qty;
            }
        }
    }
}

```

### FlexBoxTypeV subclass

```

package adproc;
/**
 * FlexBoxTypeV subclass
 */
public class FlexBoxTypeV extends FlexBox {
    /**
     * Constructor for FlexBoxTypeV subclass
     * @param dimension Array for the x,y,z dimensions of the box
     * @param sealableTop Declares whether box has a sealable top
     * @param quantity Quantity of boxes of this instance

```

```

    */
    public FlexBoxTypeV(double[] dimension, int cardGrade,
                        boolean sealableTop, int quantity){
        super(dimension, cardGrade, sealableTop, quantity);
    }
    /**
     * Method to return value of colours for TypeV FlexBox
     * @return Will be 2 for all TypeV FlexBoxes
     */
    @Override
    public int getColour(){
        return 2;
    }
    /**
     * Method to return value of reinforced bottom for TypeV FlexBox
     * @return Will be true for all TypeV FlexBoxes
     */
    @Override
    public boolean getReinforcedBottom(){
        return true;
    }
    /**
     * Method to return value of reinforced corners for TypeV
FlexBox
     * @return Will be true for all TypeV FlexBoxes
     */
    @Override
    public boolean getReinforcedCorners(){
        return true;
    }
    /**
     * Method to calculate the order price for the box(es) of this
instance
     * @return order price for the box(es) of this instance
     */
    @Override
    public double calcPrice(){
        double area = calcSurfaceArea();
        double p = area * gradePriceMultiplier[cGrade-1];

        double totalModifier = 1 + 0.15 + 0.13 + 0.12;
        p *= totalModifier;
        if(sTop == true){
            return p * 1.1 * qty;
        }else{
            return p * qty;
        }
    }
}

```



```
}
```

## BoxOrdering Class

```
package adproc;
```

```
/**
 * Class to facilitate effective creation of flexboxes
 *
 */
public class BoxOrdering {
    /**
     * Method to return a created flexbox order of a specific type
     given a flexbox's data fields
     * @param arr dimension array
     * @param grade card grade value
     * @param colours colours value
     * @param rBot reinforced bottom value
     * @param rCorn reinforced corners value
     * @param sTop sealable top value
     * @param qty quantity for this style of box
     * @return flexbox order of a specific type
     */
    public static FlexBox createFlexBoxOrder(double[] arr, int
grade, int colours, boolean rBot, boolean rCorn, boolean sTop, int
qty){

        if(
            (grade >= 1 && grade <= 3) && //conditions for a type i
box
            (colours == 0) &&
            (rBot == false) &&
            (rCorn == false)
        ){
            return new FlexBoxTypeI(arr, grade, sTop, qty);
        }else if(
            (grade >= 2 && grade <= 4) && //conditions for a
type ii box
            (colours == 1) &&
            (rBot== false) &&
            (rCorn == false)
        ){
            return new FlexBoxTypeII(arr, grade, sTop, qty);
        }else if(
            (grade >= 2 && grade <= 5) && //conditions for a
type iii box
            (colours == 2) &&
```

```

        (rBot == false) &&
        (rCorn == false)
    ){
        return new FlexBoxTypeIII(arr, grade, sTop,
qty);
    }else if(
        (grade >= 2 && grade <= 5) && //conditions for a
type iv box
        (colours == 2) &&
        (rBot == true) &&
        (rCorn == false)
    ){
        return new FlexBoxTypeIV(arr, grade, sTop, qty);
    }else if(
        (grade >= 2 && grade <= 5) && //conditions for a
type v box
        (colours == 2) &&
        (rBot == true) &&
        (rCorn == true)
    ){
        return new FlexBoxTypeV(arr, grade, sTop, qty);
    }else{
        return null;
    }
}
}
}

```

# Coursework Specification

ADPROC, Advanced Programming Concepts (U21266)

## Coursework

**Hand out date: 24.X.2018 Hand in date: 7.XII.2018 (Demonstration: by week starting 3<sup>rd</sup> Dec)**

This is an assessed piece of group coursework, it is therefore essential to be completed and handed-in on time. If you are unclear about any aspect of the assignment, including the assessment criteria, please raise this at the first opportunity. The usual regulations apply to a late submission of work. The submitted application must be in Java (using Java NetBeans IDE) to be marked. During the demonstration (by week 12 – week starting 3<sup>rd</sup> December, in your lab session) **you have to submit a memory stick** with your source code and Java NetBeans project files with **your group number** on it. **The coursework you submit should be your group work. If your coursework includes other people's ideas and material, they must be properly referenced or acknowledged. Failing to do so intentionally or unintentionally constitutes plagiarism. The University treats plagiarism as a serious offence.**

## ORDER SYSTEM FOR A BOX-SELLING COMPANY

The Chinese invented cardboard in the 1600s and the English created the first commercial cardboard box in 1817. “FlexBox” is a company producing variety of boxes for packaging wide range of goods. Due to the wide range of requirements of their customers, the variety of boxes the FlexBox has to produce is very extensive.



The boxes are all rectangular and have the following characteristics:

- They are all made of cardboard;
- The cardboard has a specified grade;
- The boxes may have no printing, or 1, or 2 colour printing;
- Some boxes may have reinforced bottoms;
- Some boxes may have reinforced corners;
- All boxes may have sealable tops.

**Table 1.** Types of cardboard boxes available.

Type	Grade of cardboard	Colour print			Reinforced bottom	Reinforced corners
		0	1	2		
I	1 – 4	YES	NO	NO	NO	NO
II	2 – 4	NO	YES	NO	NO	NO
III	2 – 5	NO	NO	YES	NO	NO
IV	2 – 5	NO	NO	YES	YES	NO
V	3 – 5	NO	NO	YES	YES	YES

**Fig. 1.** Simple cardboard boxes.

The types of boxes, produced by the company, are shown in Table1 and the costs of 1m<sup>2</sup> of cardboard are given in Table2.

**Table 2.** Basic cost of 1 square metre of cardboard.

Cardboard Grade	1	2	3	4	5
Cost per m <sup>2</sup> [in £]	0.55	0.65	0.82	0.98	1.5

**Table 3.** Additional costs.

1 colour	12% extra
2 colours	15% extra
Reinforced bottom	13% extra
Reinforced corners	12% extra
Sealable tops	10% extra

There are some additional costs depending on whether the box has printing and if there is any box reinforcement. These are shown in Table 3 and the percentage increases **are all applied using the basic cost**. All boxes may have sealable tops.

When a customer asks FlexBox to quote a price for an order, they specify the following:

- The size of the box (width, length, and height);
- The grade of the cardboard;
- Whether they want any colour printing (no colour, or 1, or 2 colour printing);
- Whether they want any bottom and/or corner reinforcement;
- Whether the box has a sealable top;
- The quantity of boxes.

From this information, the order system should determine if the type of the requested box can be supplied by FlexBox, if it cannot, it should display an appropriate message and reject the order. If the ordered box/boxes correspond to any of the types given in Table 1, and can be supplied by FlexBox, the cost of the order must be calculated (using Table 2 and Table 3) and quoted.

The customer should be able to place several orders in one session, in which case the total cost should be prompted.

**Customers should not be asked for the type of box they want** (since this is only used within the company to calculate the cost). **It is your application that must determine (using Table 1) the box type, based on the ordered box attributes.**

Customers should be able to get a quote for as many pipes (of different types) as they like (within the capacity of *FlexBox*) in the same order. In such cases, the total cost of the order should be calculated and displayed.

Your user interface should be a GUI (graphical user interface) using AWT/Swing. If no GUI is developed, you will lose the marks allocated for this part of your coursework.

## Your Task

- Write an application, which will allow the customers to enter the details of their order and subsequently prompts the cost of the order. Your application should verify that *FlexBox* can supply the type of the requested boxes (the customer should not be asked to specify the box type).
- Use OO design approach (abstraction, inheritance, aggregation, and polymorphism) and create a class hierarchy that describes the types of boxes *FlexBox* sells. Use an abstract class as well.
- Give UML use case diagram, UML class hierarchy diagram, one class and one instance diagrams.
- Use proper level of abstraction, encapsulation and accessibility for the class attributes and methods. Application with no levels of abstraction will fail the coursework!
- Devise suitable test plan and data, which you can use to test the performance of your ordering system.

## Assessment Criteria

You should give **a demonstration and submit a memory stick** (with **your group number on it**) with your source code and Java NetBeans project files of your software no later than week12 (week starting **3.XII.2018**), during your lab session.

On **7.XII.2018** your group should submit electronically (**by 6pm**) to Moodle a **.pdf** file with your **report**. **The file name should be your group name** (e.g., *GrC-2.pdf*, or *GrA-3.pdf*, or *GrD-5.pdf*, etc.). **Only one file per group** should be submitted (decide in advance who is going to submit it). Your report **should be no more than 6 pages (excluding the appendices)** and should include the following:

- ✓ **A UML** use case diagram of your order system, UML class hierarchy diagram of your OO application design, and also one UML class diagram (one class of your choice), and one instance diagram;
- ✓ **A brief** description of the application including any assumptions you have made and any limitations in your implementation of the application;
- ✓ **A test** schedule (no more than one page) and screen shots to evidence the testing and evaluation;
- ✓ **The source code** that you have written as an Appendix (the same code that you used in your demonstration);
- ✓ **Some sample** input and output (screenshots) to demonstrate your application is working;
- ✓ **A Group contribution form** with your individual contributions;
- ✓ **This document.**

The assessment criteria and allocated marks are given in Table 4.

**Table 4.** Assessment criteria and marks distribution.

Topic/Criteria	Comments	Marks available	Marks awarded
Class hierarchy descriptions (UML)	How suitable is the design and the adopted hierarchy of the application? Use of abstract class?	10 (Report)	
UML class and instance diagrams	Are the UML use case, class and instance diagrams relevant to the application?	10 (Report)	
Code and functionality	How complete is the implementation? Does it perform as specified? Does it use an OO design approach? Use of abstract class? Are the class attributes and methods at the appropriate hierarchy level? Is the verification and validation of input data adequate? Is the exception handling properly done? Are the style, indentation and comments appropriate? Is the layout clear?	45 (Demo(20), Report(25))	
Using AWT/Swing	Is the layout clear? How well designed is the interface? How appropriate is the use of components? How appropriate is the use of attributes? Is it working, or just an attempt?	15 (Demo)	
Testing	How thorough is planning and testing? Does it cover most/few possible errors?	10 (Report)	
Supporting documentation and comments.	Is the text clearly written and well presented? Are the assumptions, limitations, problems and features of the application well documented?	10 (Report)	
OVERALL MARK		100	

Dr. I. Jordanov

2 