

4 Decision making in programs

Many computer programs require decisions to be made, depending on the data entered. In this chapter we will develop some programs in which choices have to be made during data processing.

The first example asks you to write a program to decide the grades which will be awarded to students in their course assessment:

Students' work is assessed at the end of the first year of a two year course, and is awarded a grade:

- 70% or over is a Distinction
- 60% or over, but less than 70%, is a Merit
- 40% or over, but less than 60%, is a Pass
- less than 40% is a Fail

Students who fail are allowed to resubmit their work, and must achieve a Pass grade before continuing to the second year of the course. However, the mark awarded for the second attempt will be limited to a maximum of 40%, which is a bare pass.

A program is required which will input:

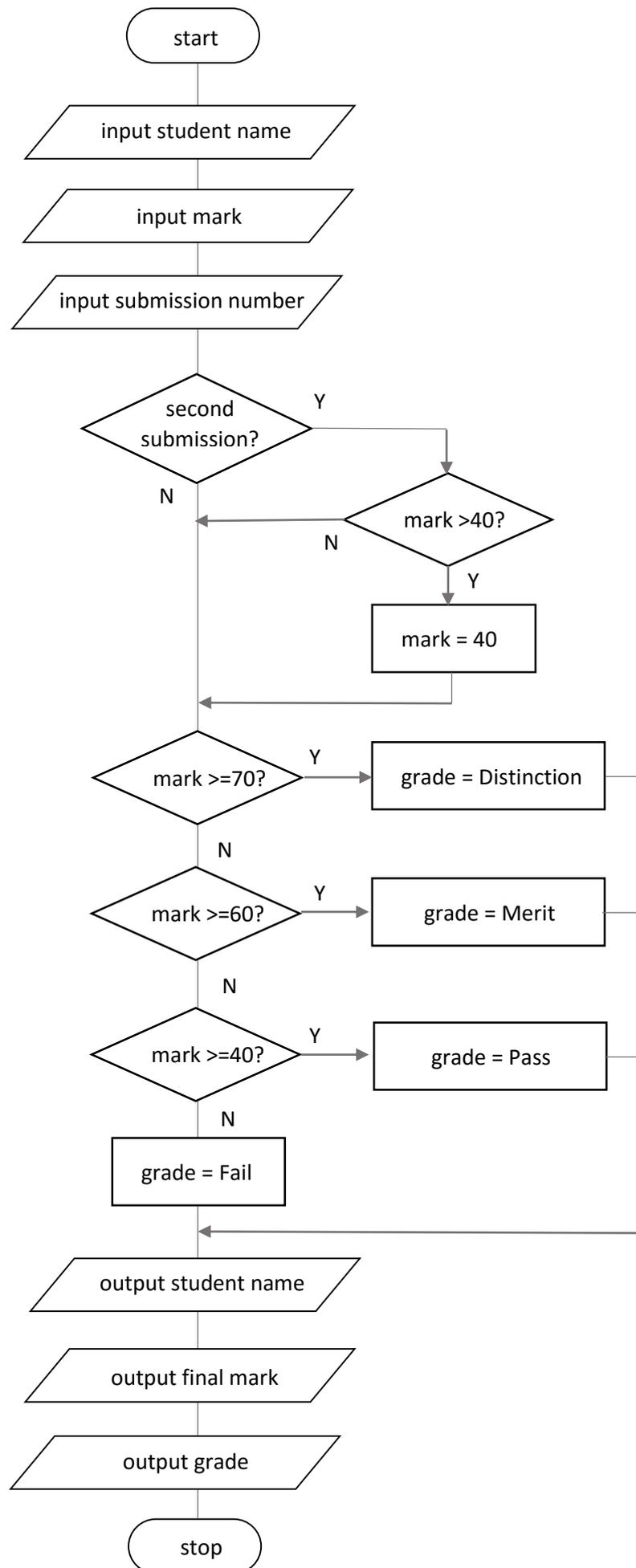
- the student's name
- the mark awarded
- whether this is a first or second submission

The program should then output:

- the student's name
- the mark awarded, restricted to a maximum of 40% for a second submission
- the grade awarded.

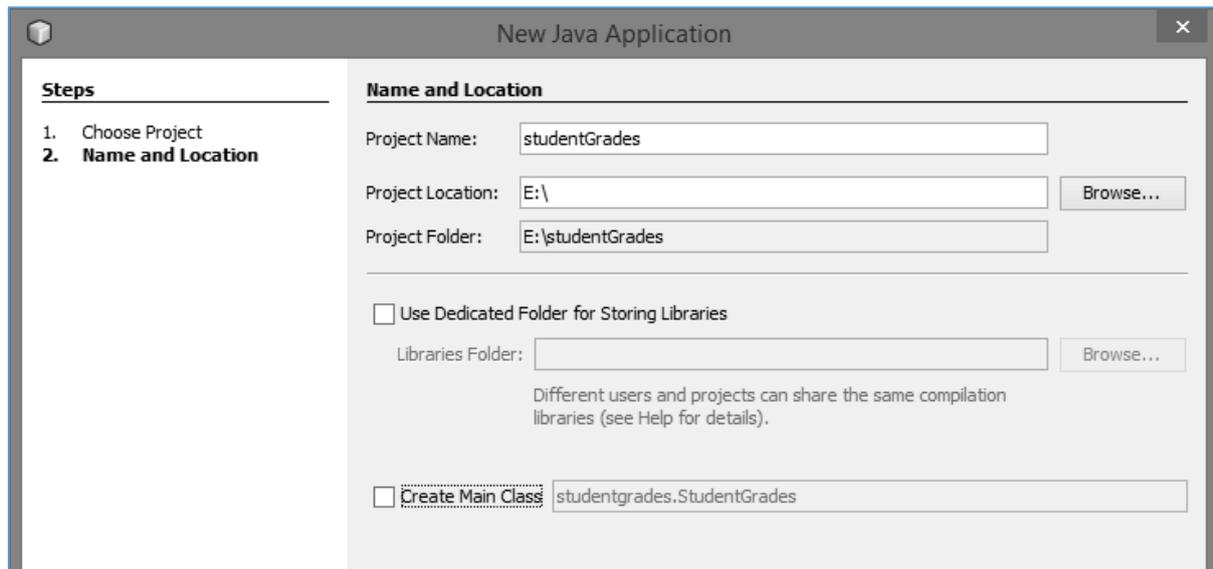
This program involves some complex processing, so it is a good plan to begin by producing a flowchart. A possible design is shown on the next page:

- We begin by inputting the student's name, percentage mark, and whether this is the first or second submission of their work.
- In the case of a second submission, any mark above 40% must be reduced to 40% as a penalty for requiring two attempts.
- The program then checks the mark and decides the corresponding grade.
- Finally, the required information is output, with the mark adjusted if necessary for a second submission.

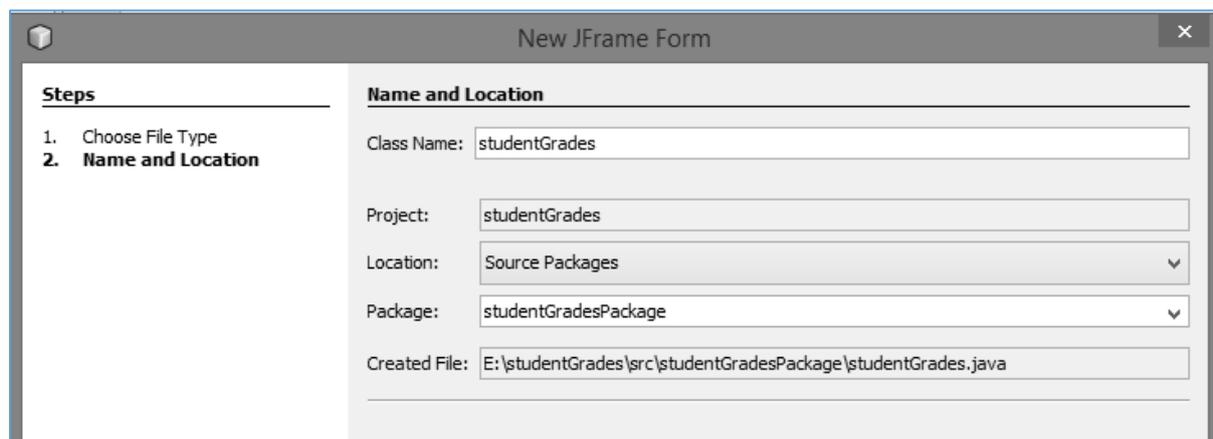


We can now begin the programming.

Close all projects, then set up a new project. Give this the name ***studentGrades***, and ensure that the ***Create Main Class*** option is not selected:



Click the ***Finish*** button to go to the NetBeans editing page. Right-click on the ***studentGrades*** project, and select ***New / JFrame Form***. Give the ***Class Name*** as ***studentGrades***, and the ***Package*** as ***studentGradesPackage***:

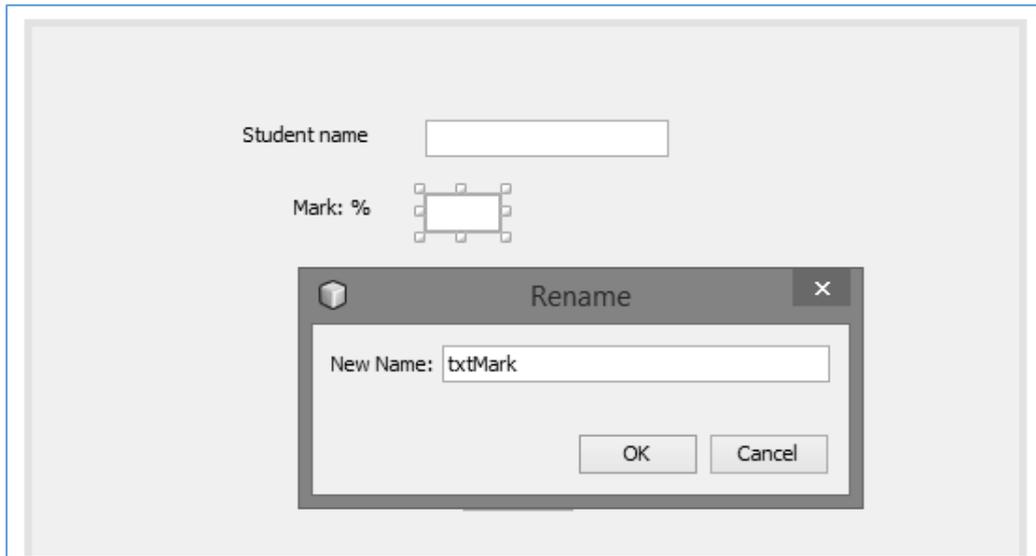


Click the ***Finish*** button to return to the NetBeans editing screen.

- Right-click on the ***form***, and select ***Set layout / Absolute layout***.
- Go to the ***Properties*** window on the bottom right of the screen and click the ***Code*** tab. Select the option: ***Form Size Policy / Generate pack() / Generate Resize code***.
- Click the ***Source*** tab above the design window to open the program code. Locate the main method. Use the + icon to open the program lines and change the parameter "***Nimbus***" to "***Windows***".

Run the program and accept the ***main*** class which is offered. Check that a blank window appears and has the correct size and colour scheme. Close the program window to return to the NetBeans editing screen.

Begin by placing **labels** and **text fields** on the form for entering the student's name and mark. Rename the text fields as **txtStudentName** and **txtMark**:



We can use a drop-down list for selecting **first** or **second** submission of the student's work. Locate the **Combo Box** component in the **palette**, then drag and drop this on the form. Rename the combo box as **cmbSubmission**.

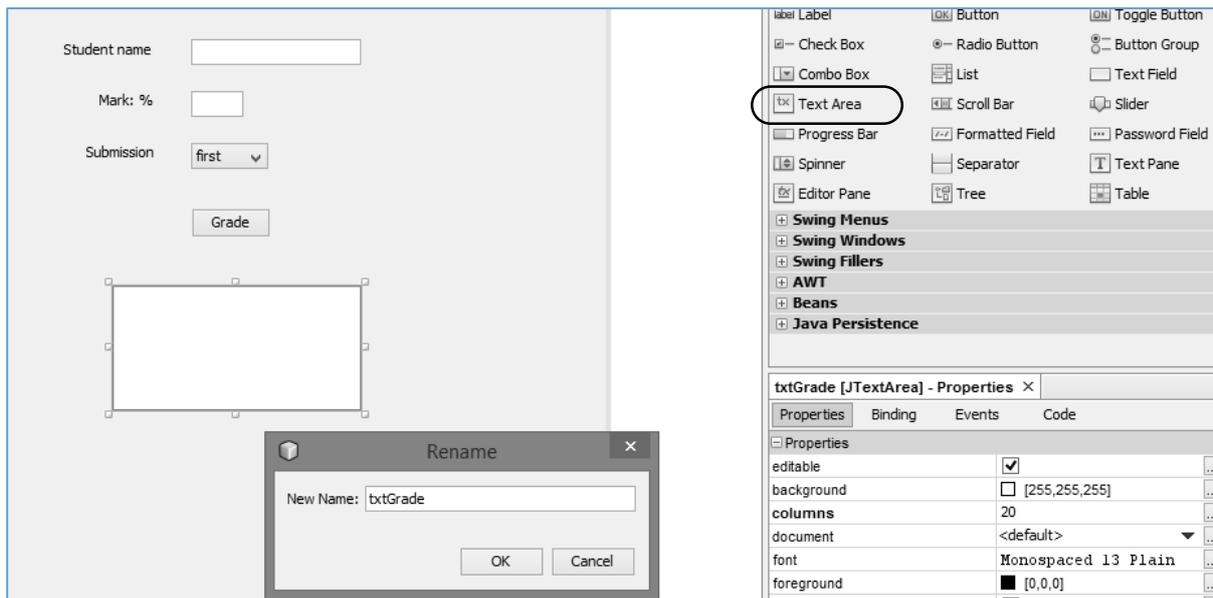
Go to the **model** property and enter the words "**first**" and "**second**", separated by a comma.

Drag and drop a **label** to the left of the combo box, and set the caption to "**submission**":

The screenshot shows the Java Swing IDE. The form now has three components: 'Student name', 'Mark: %', and 'Submission' (a label next to a dropdown menu showing 'first'). A 'Rename' dialog box is open, showing 'New Name: cmbSubmission'. The Properties window on the right shows the properties for 'cmbSubmission [JComboBox]'. The 'model' property is set to 'first, second'.

cmbSubmission [JComboBox] - Properties			
Properties	Binding	Events	Code
background	<input type="checkbox"/>	[255,255,255]	
editable	<input type="checkbox"/>		
font		Tahoma 11 Plain	
foreground	<input checked="" type="checkbox"/>	[0,0,0]	
maximumRowCount		8	
model		first, second	
selectedIndex		0	

Complete the form by adding a **button** and **text area**. Change the button caption to “**Grade**”, and give this the name **btnGrade**. Rename the text area as **txtGrade**:



We can now begin the program code.

Double click the **Grade** button to create a method. Add **local variables** to store the student mark, first or second submission, and the grade description awarded.

It will be helpful to display an error message box if the student mark is not entered in a correct integer format. Add a **try ... catch** block to do this:

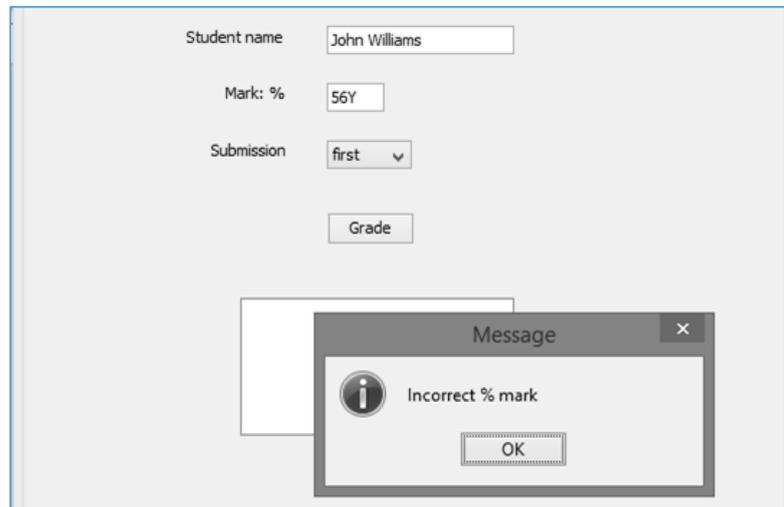
```
private void btnGradeActionPerformed(java.awt.event.ActionEvent evt) {
    int studentMark;
    String grade;
    String submission;

    try
    {
        studentMark=Integer.parseInt(txtMark.getText());
    }
    catch(NumberFormatException e)
    {
        JOptionPane.showMessageDialog(studentGrades.this, "Incorrect % mark");
    }
}
```

Go to the start of the program and add the code to create the message box:

```
package studentGradesPackage;
import javax.swing.JOptionPane;
public class studentGrades extends javax.swing.JFrame {
```

Run the program. Enter a correct integer number as the student mark and click the Grade button. This should be accepted by the program. Now enter a number in an incorrect format and the error message box should be displayed:

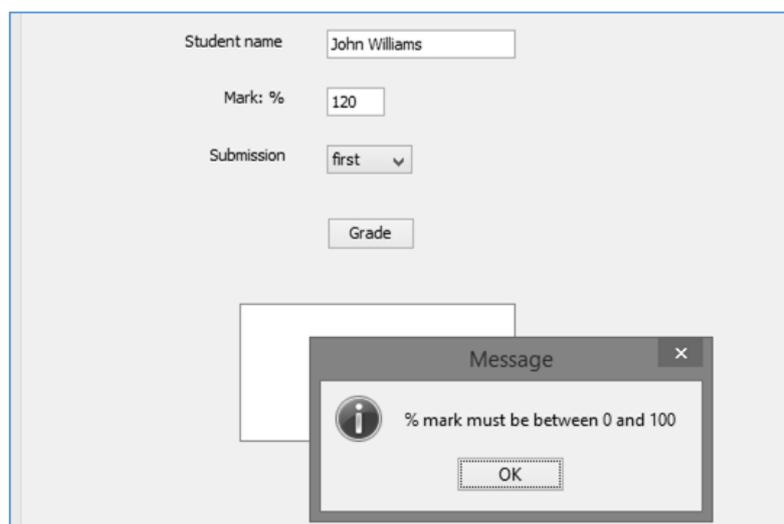


Close the program and return to the editing screen. Another possible input error is that the mark entered is outside the range for a percentage. We can add a range check as validation. Notice the double vertical line symbol “ || ” which provides the logical operation “ OR ”:

```
try
{
    studentMark=Integer.parseInt(txtMark.getText());

    if (studentMark<0 || studentMark>100)
    {
        JOptionPane.showMessageDialog(studentGrades.this,
                                     "% mark must be between 0 and 100");
    }
}
catch(NumberFormatException e)
{
```

Run the program and enter a mark which is outside the percentage range 0 – 100. An error message should be displayed:



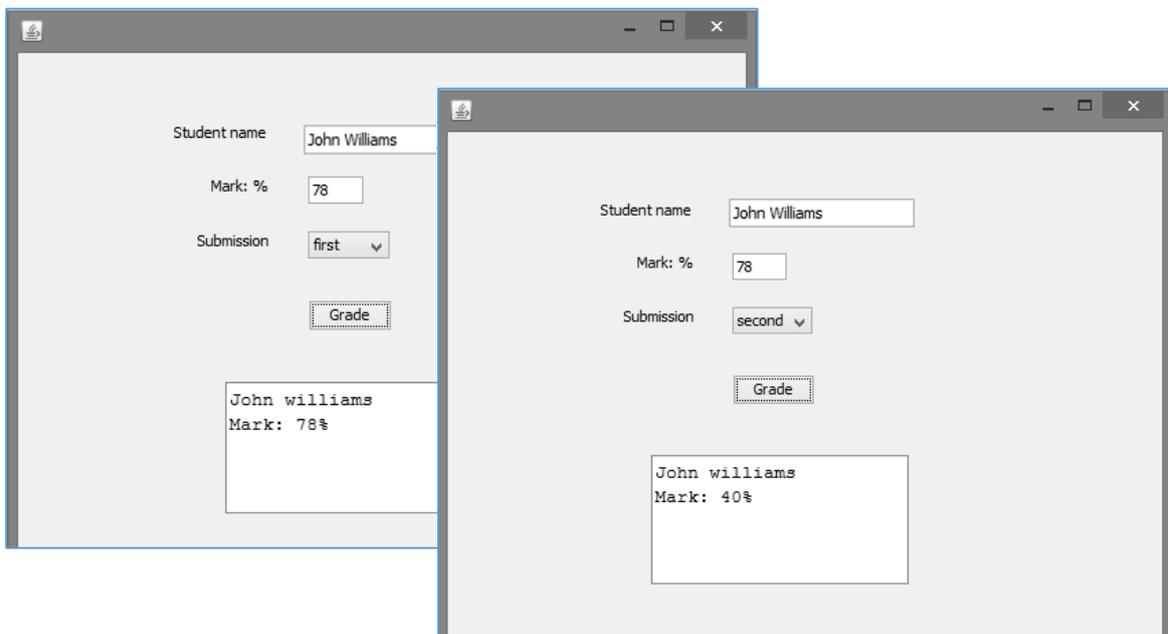
Close the program and return to the editing screen.

The next stage of the algorithm is to check for a second submission of the student's work. In this case, the mark that is awarded must be limited to a maximum of 40%. Add code to carry out this check.

We will also start to build up a text string for display of the results in the **text area**:

```
try
{
    studentMark=Integer.parseInt(txtMark.getText());
    if (studentMark<0 || studentMark>100)
    {
        JOptionPane.showMessageDialog(studentGrades.this,
            "% mark must be between 0 and 100");
    }
}
else
{
    String s="";
    s += txtStudentName.getText();
    submission = (String)cmbSubmission.getSelectedItemAt();
    if (submission.equals("second"))
    {
        if (studentMark>40)
        {
            studentMark=40;
        }
    }
    s += "\nMark: "+Integer.toString(studentMark)+"%";
    txtGrade.setText(s);
}
}
catch(NumberFormatException e)
{
```

Run the program. Check that student names and marks are displayed correctly, with the mark limited to 40% in the case of a second submission:



Close the program and return to the editing screen. The final stage is to decide the grade that will be awarded. Add conditional **IF ... THEN ... ELSE** statements to do this.

```
        if (submission.equals("second"))
        {
            if (studentMark>40)
            {
                studentMark=40;
            }
        }
        s += "\nMark: "+Integer.toString(studentMark)+"%";

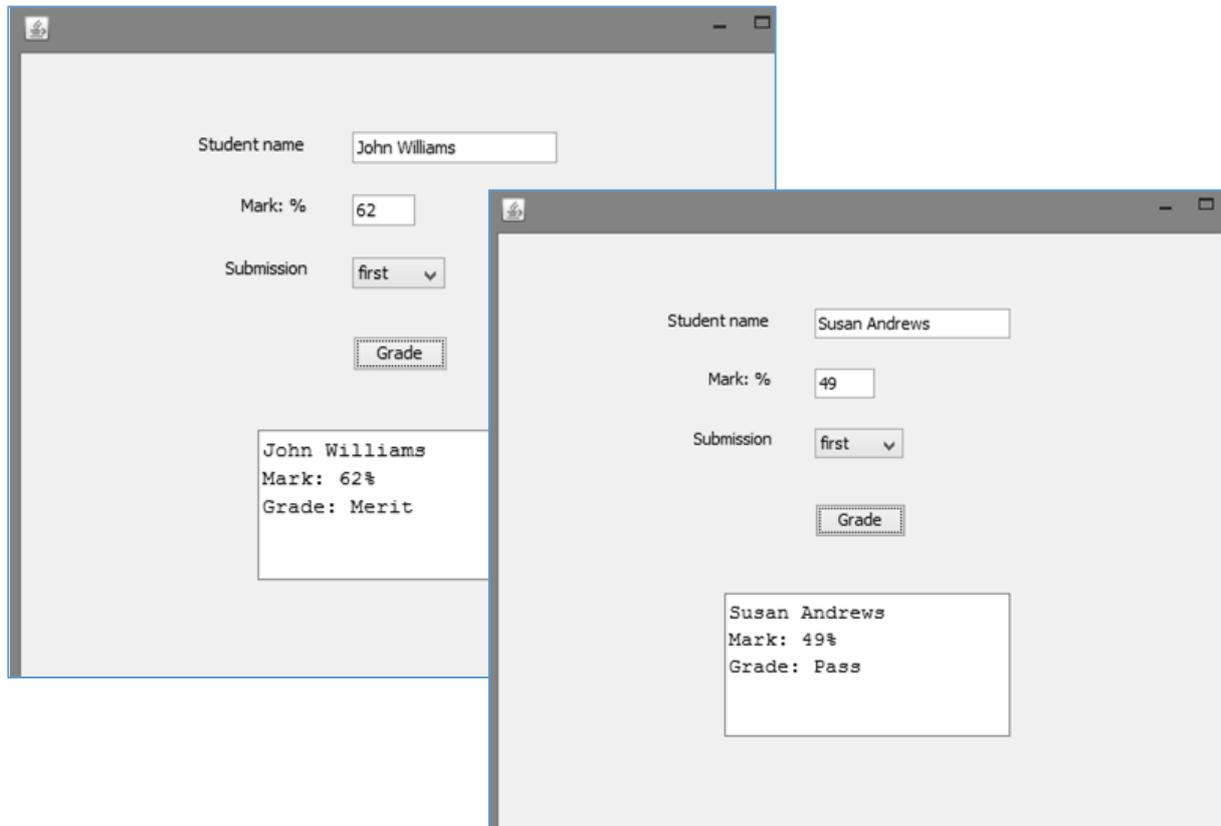
        if (studentMark>=70)
        {
            grade="Distinction";
        }
        else
        {
            if (studentMark>=60)
            {
                grade="Merit";
            }
            else
            {
                if (studentMark>=40)
                {
                    grade="Pass";
                }
                else
                {
                    grade="Fail";
                }
            }
        }
        s += "\nGrade: "+ grade;

        txtGrade.setText(s);
    }
}
catch(NumberFormatException e)
{
```

Notice that **indenting** has been used in the IF ... THEN ... ELSE blocks. After each opening bracket, the following lines of code are moved to the right by about three spaces. When the end of the conditional block is reached, the closing bracket is moved back to the same indent position as the opening bracket.

Using indentation correctly can make the structure of the program easier to understand, and reduces the chances of an error due to an incorrect number of closing brackets.

Run the program. Check that correct grades are awarded for a range of different marks:



Close the program and return to the NetBeans editing page.

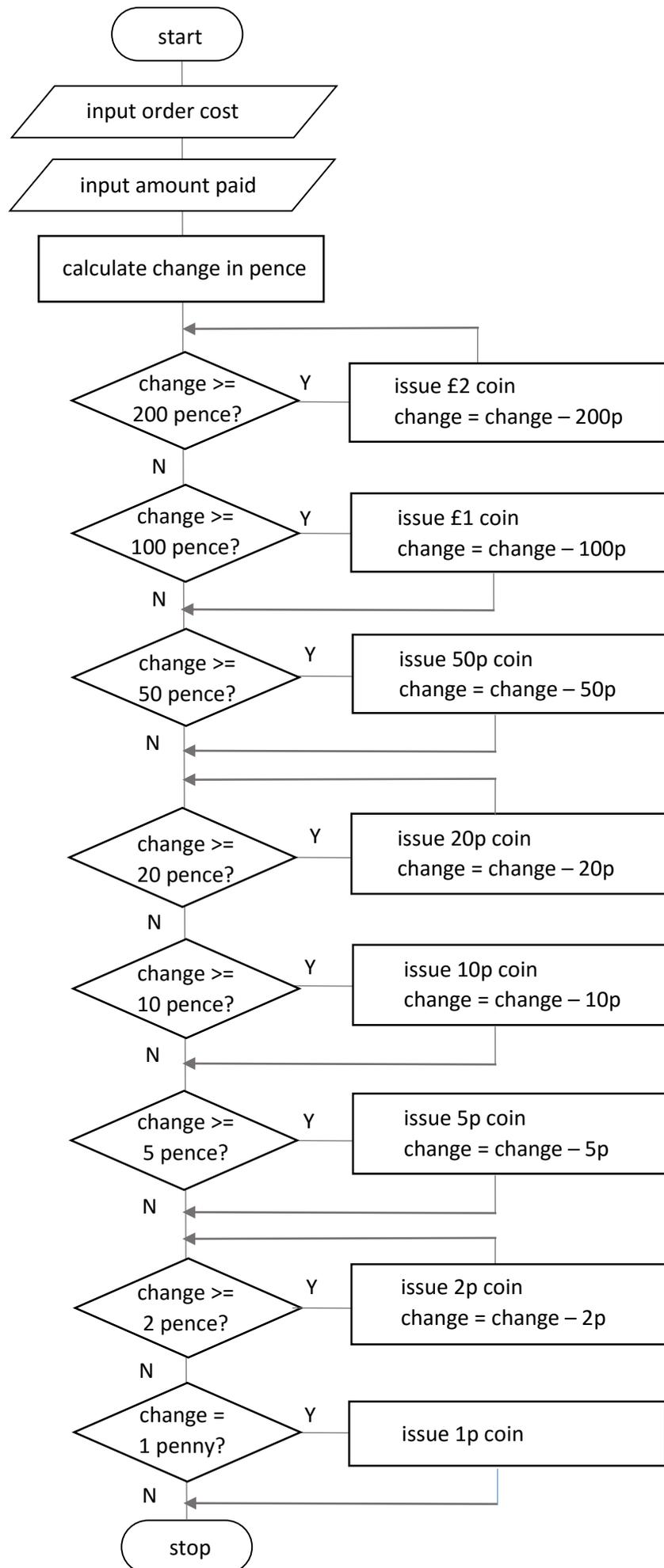
For our next project we will produce a program to calculate the change to be given to customers when they make payment in a shop:

A program is required which will specify the coins which should be given in change, up to a maximum of £5.00, when payment is made for purchases. The minimum number of coins should be given.
 Coins may be chosen from: 1 penny, 2 pence, 5 pence, 10 pence, 20 pence, 50 pence, 1 pound, 2 pounds.

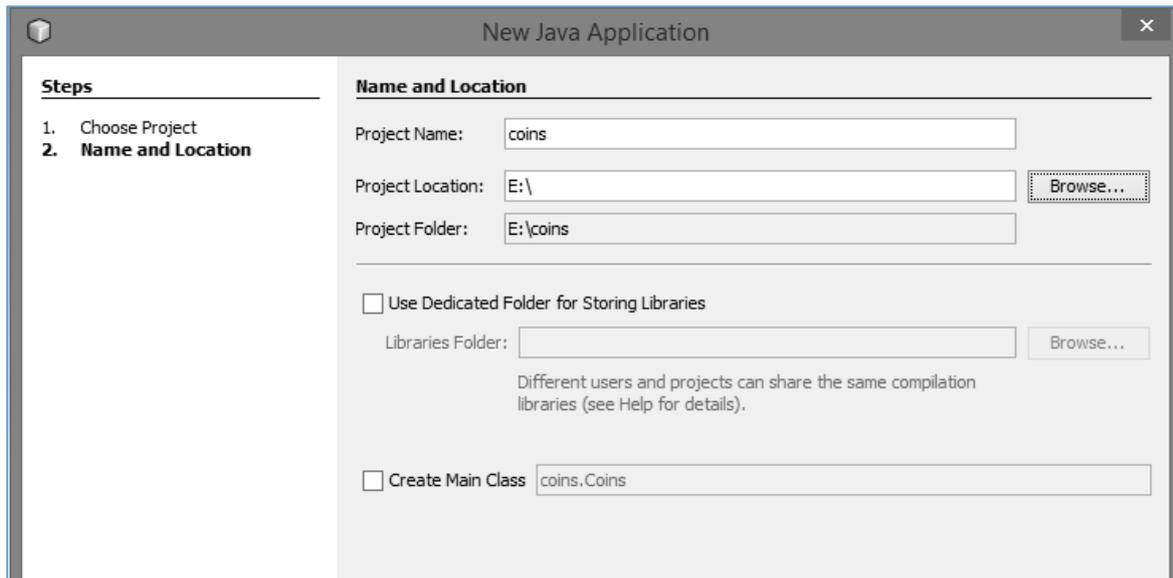
This is a slightly more difficult problem than it first seems:

- Some coins can only be given in change once – for example, two 5p coins could be replaced by a 10p coin.
- Some coins may have to be given twice – for example, 40p change can only be given as two 20p coins.

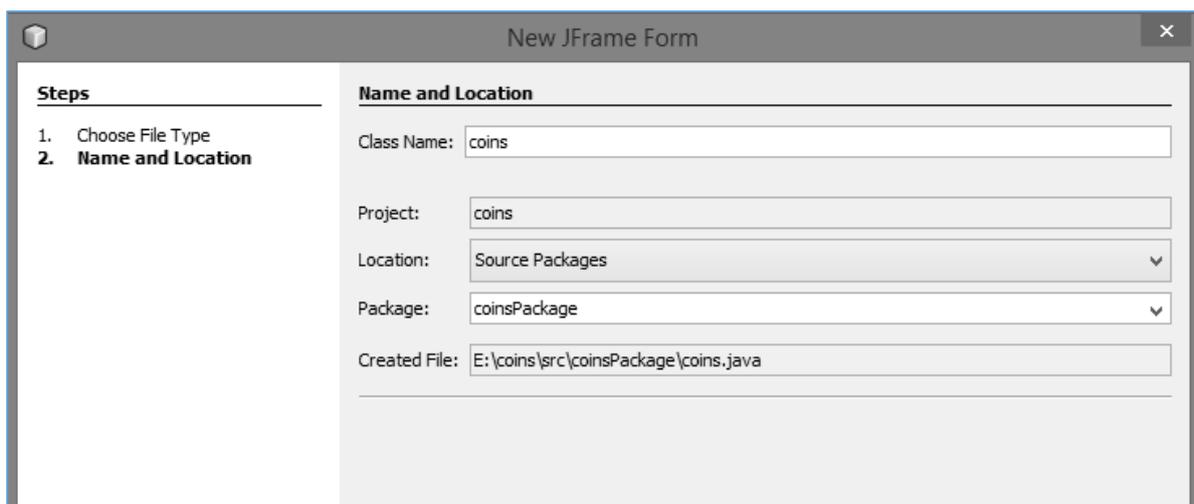
A possible design for the algorithm is given in the flow chart on the next page. The overall strategy is to find the change due in pence, then reduce the amount of pence remaining as each possible coin is issued as change. By starting with the highest value coin, the minimum number of coins should be given to the customer as change.



Close all projects, then set up a new project. Give this the name *coins*, and ensure that the **Create Main Class** option is not selected:



Click the **Finish** button to return to the NetBeans editing page. Right-click on the *coins* project, and select **New / JFrame Form**. Give the **Class Name** as *coins*, and the **Package** as *coinsPackage*:



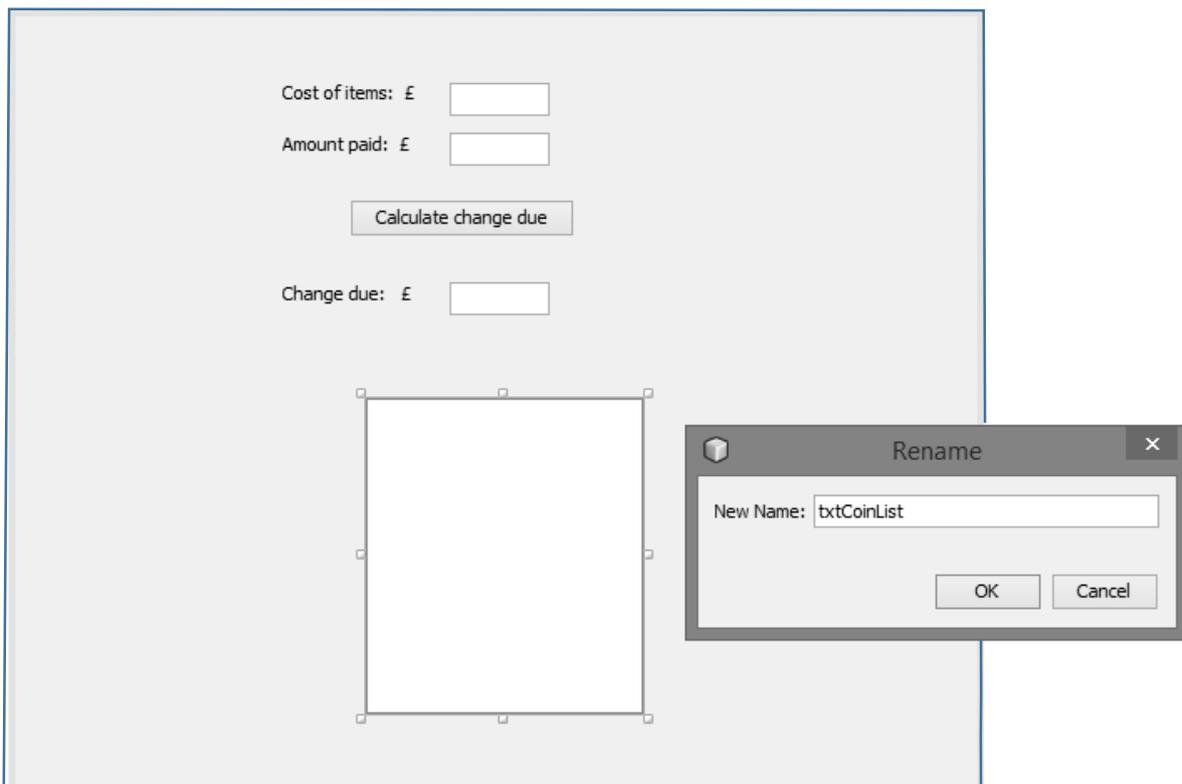
Return to the NetBeans editing screen.

- Right-click on the *form*, and select **Set layout / Absolute layout**.
- Go to the **Properties** window on the bottom right of the screen and click the **Code** tab. Select the option: **Form Size Policy / Generate pack() / Generate Resize code**.
- Click the Source tab above the design window to open the program code. Locate the main method. Use the + icon to open the program lines and change the parameter "**Nimbus**" to "**Windows**".

Run the program and accept the *main* class which is offered. Check that a blank window appears and has the correct size and colour scheme.

Close the program and return to the editing screen. Click the **Design** tab to open the form layout view.

- Add **labels** and **text fields** for the cost of the items purchased and the amount paid by the customer. Rename the text fields as **txtTotalCost** and **txtAmountPaid**.
- Add a **button** with the caption “**Calculate change due**”, and rename this as **btnCalculateChange**.
- Add a **label** and **text field** to display the change due. Rename the text field as **txtChangeDue**.
- Finally, add a text area for display of the coins to be issued as change. Name this as **txtCoinList**:



Double click the “**Calculate change due**” button to create a **method**. Add **local variables** to store amounts of money, and a **try ... catch** block for error trapping if an incorrect number is entered.

```
private void btnCalculateChangeActionPerformed(java.awt.event.ActionEvent evt) {
    double totalCost, amountPaid, changeDue;
    int change=0;

    try
    {
        totalCost=Double.parseDouble(txtTotalCost.getText());
        amountPaid=Double.parseDouble(txtAmountPaid.getText());
    }
    catch(NumberFormatException e)
    {
        JOptionPane.showMessageDialog(coins.this, "Incorrect number format");
    }
}
```

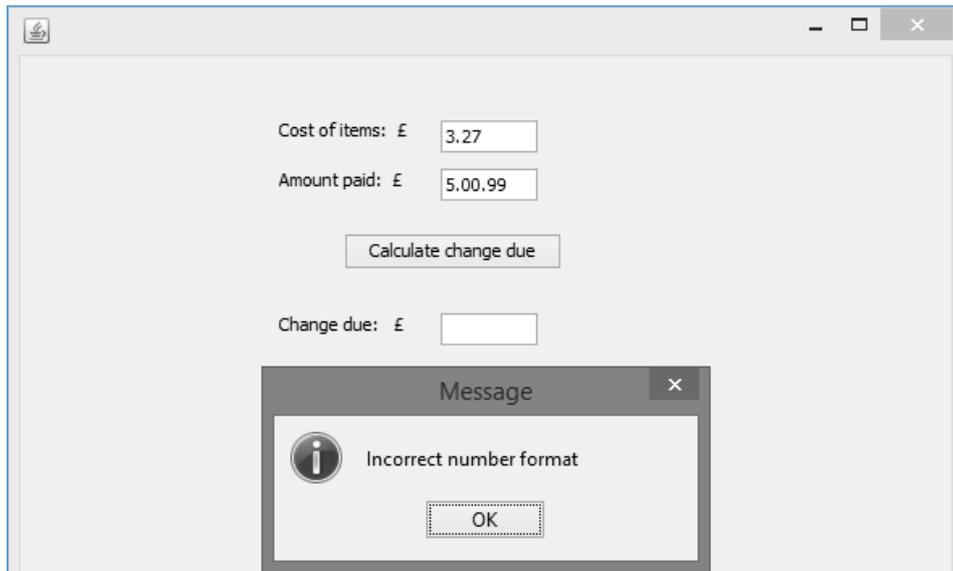
Go to the start of the program and add the code to create the message box:

```
package coinsPackage;

import javax.swing.JOptionPane;

public class coins extends javax.swing.JFrame {
```

Run the program and check that incorrect number input is detected:



Close the program and return to the editing screen.

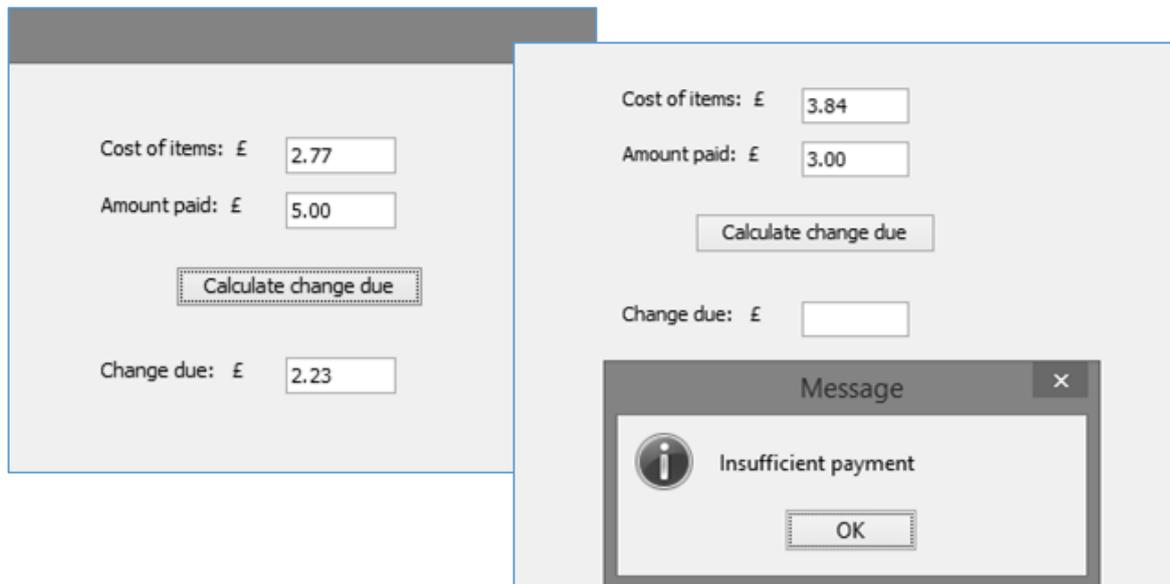
We should now check that the customer has paid *sufficient* money for the items purchased, and give an error message if this is not the case. Add the lines of code below:

```
try
{
    totalCost=Double.parseDouble(txtTotalCost.getText());
    amountPaid=Double.parseDouble(txtAmountPaid.getText());

    changeDue=amountPaid-totalCost;

    if (changeDue<0)
    {
        JOptionPane.showMessageDialog(coins.this, "Insufficient payment");
    }
    else
    {
        txtChangeDue.setText(String.format("%.2f", changeDue));
    }
}
catch(NumberFormatException e)
{
    JOptionPane.showMessageDialog(coins.this, "Incorrect number format");
}
}
```

Run the program. Test the cases where sufficient money *has* or *has not* been paid:



Close the program and return to the editing screen.

We can now start work on the calculation algorithm:

- Begin by converting the amount of change due into an integer number of pence.
- The highest value coin available is £2. There is a possibility that two of these coins will be needed in change, so a WHILE... loop will be used. A coin will be dispensed each time around the loop until the remaining change is less than 200 pence.
- The next coin is £1. In this case, a maximum of one coin could be included in the change, so no loop is needed. Instead, we will use an IF... condition:

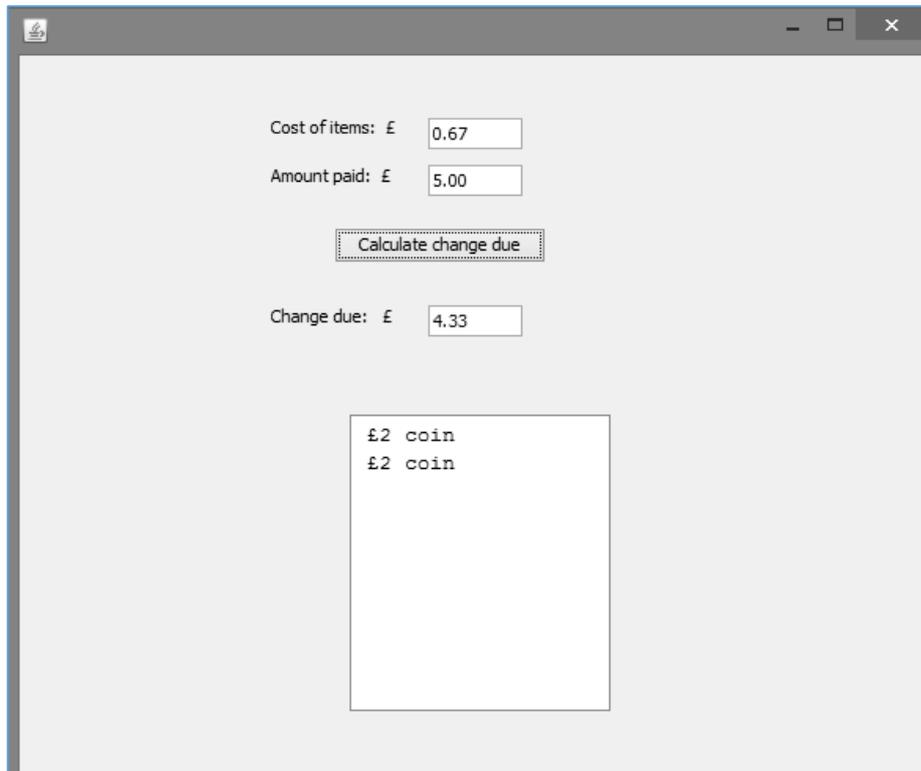
```

if (changeDue<0)
{
    JOptionPane.showMessageDialog(coins.this, "Insufficient payment");
}
else
{
    txtChangeDue.setText(String.format("%.2f", changeDue));

    change=(int) Math.round(changeDue*100);
    String s="";
    while(change>=200)
    {
        s += " £2 coin \n";
        change -= 200;
    }
    if (change>=100)
    {
        s += " £1 coin \n";
        change -= 100;
    }
    txtCoinList.setText(s);
}
}
catch(NumberFormatException e)
{

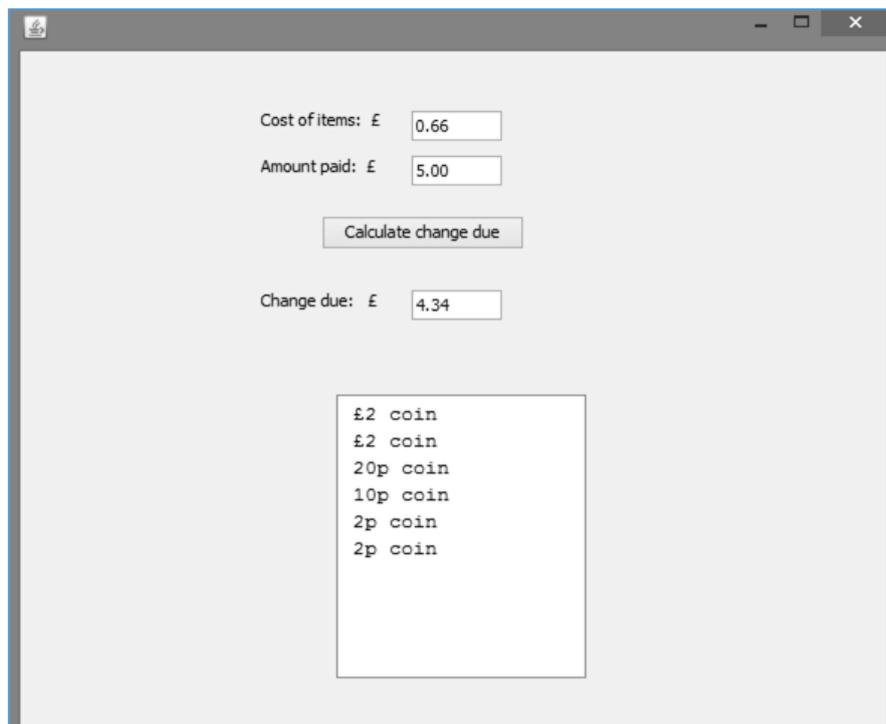
```

Run the program. Test that the correct combination of £2 and £1 coins are given for different amounts of change due:



Close the program and return to the editing screen. Add the blocks of code to issue the remaining coins: **50p, 20p, 10p, 5p, 2p, 1p**, as shown on the next page.

Run the program. Carry out tests for purchases requiring various amounts of change up to £5. Confirm that in each case the amount of change is correct, and that the minimum number of coins have been used:



```
if (changeDue<0)
{
    JOptionPane.showMessageDialog(coins.this, "Insufficient payment");
}
else
{
    txtChangeDue.setText(String.format("%.2f", changeDue));
    change=(int) Math.round(changeDue*100);
    String s="";
    while(change>=200)
    {
        s += " £2 coin \n";
        change -= 200;
    }
    if (change>=100)
    {
        s += " £1 coin \n";
        change -= 100;
    }

    if (change>=50)
    {
        s += " 50p coin \n";
        change -= 50;
    }
    while (change>=20)
    {
        s += " 20p coin \n";
        change -= 20;
    }
    if (change>=10)
    {
        s += " 10p coin \n";
        change -= 10;
    }
    if (change>=5)
    {
        s += " 5p coin \n";
        change -= 5;
    }
    while (change>=2)
    {
        s += " 2p coin \n";
        change -= 2;
    }
    if (change==1)
    {
        s += " 1p coin \n";
    }

    txtCoinList.setText(s);
}
}
catch(NumberFormatException e)
```

The final program in this chapter asks you to carry out a common calculation made by doctors and nurses:

A program is required which will calculate Body Mass Index (BMI). This is a value used by health professionals to assess whether a person has a healthy weight for their height, or is overweight or underweight.

The weight and height of a person is to be entered either in metric or imperial units (kilograms and metres, or stones/pounds and feet/inches), along with gender.

Body mass index is calculated using the formula:

$$BMI = \frac{\text{weight (kilograms)}}{[\text{height(metres)}]^2}$$

The program should display the Body Mass Index value, and provide a message to indicate whether the person's weight is within the healthy range, overweight or underweight.

BMI ranges	Male	Female
Underweight	< 20	< 19
Healthy range	20 – 25	19-24
Overweight	> 25	> 24

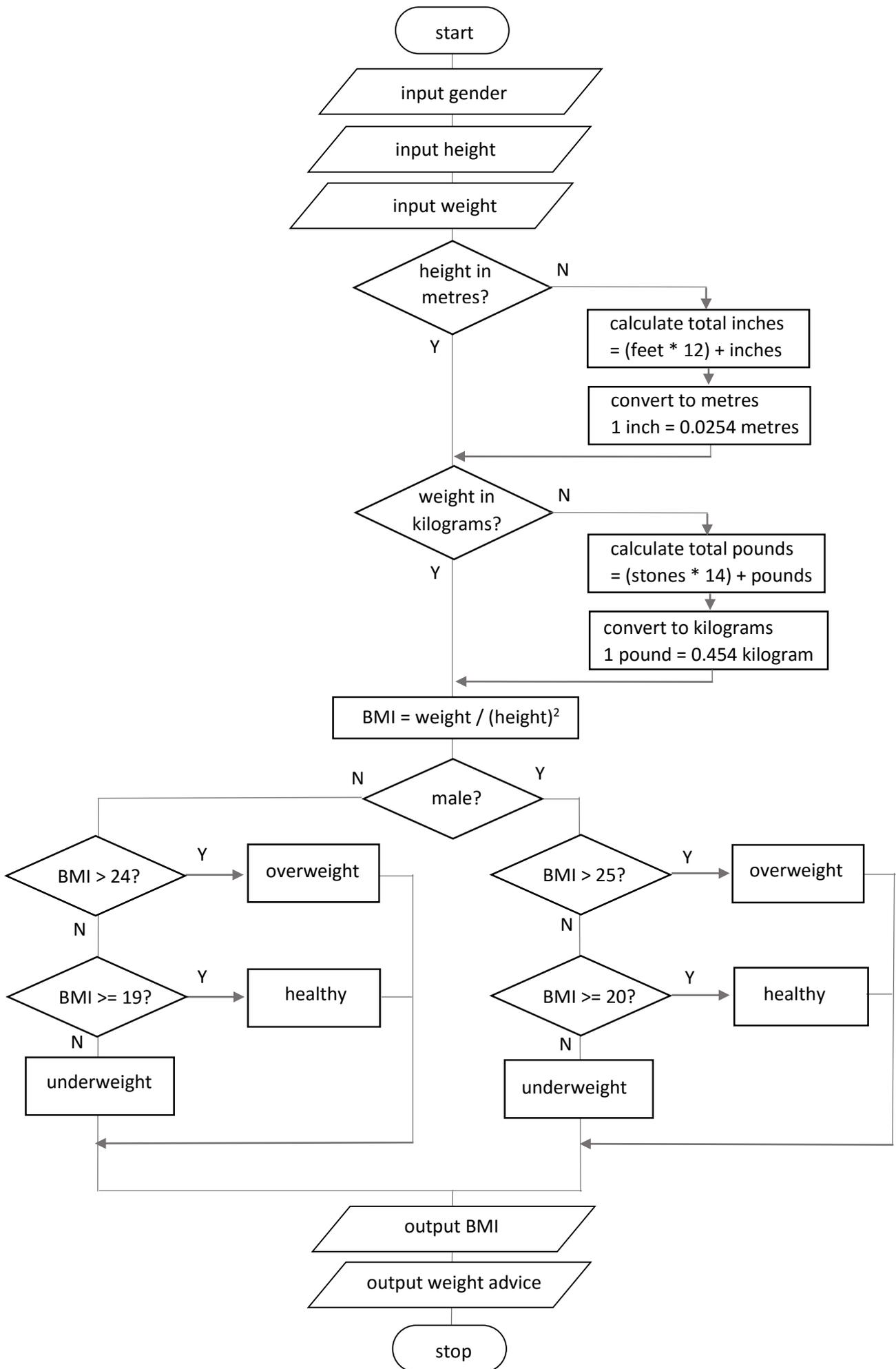
The input values required for the calculation will be the height and weight of the person, and their gender.

Body mass index is calculated using *metric* units of *metres* and *kilograms*. We must provide an option for the heights and weights be entered as *feet/inches* and *stones/pounds* in the *imperial* system, then carry out a conversion:

1 inch = 0.0254 metres

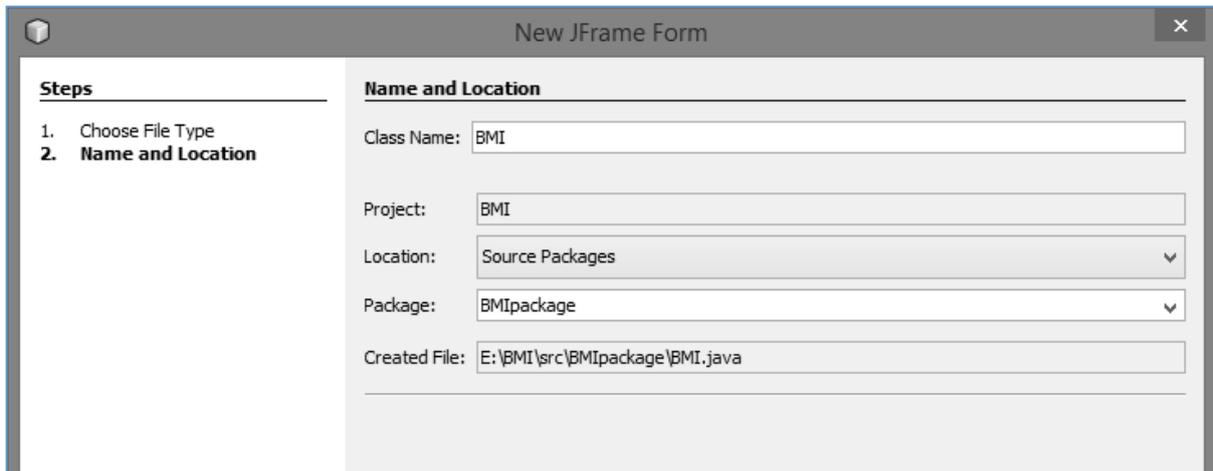
1 pound = 0.454 kilogram

A possible algorithm for the program is given in the flowchart below.



Close all projects, then set up a **New Project**. Give this the name **BMI**, and ensure that the **Create Main Class** option is not selected.

Return to the NetBeans editing page. Right-click on the **BMI** project, and select **New / JFrame Form**. Give the **Class Name** as **BMI**, and the **Package** as **BMIpackage**:



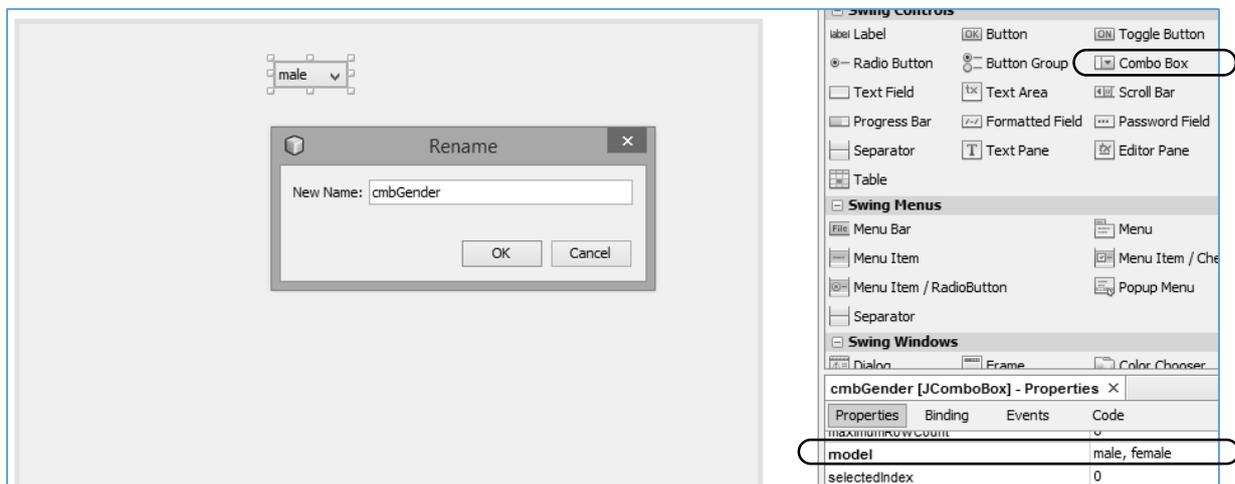
Click the **Finish** button to return to the NetBeans editing screen.

- Right-click on the **form**, and select **Set layout / Absolute layout**.
- Go to the **Properties** window on the bottom right of the screen and click the **Code** tab. Select the option: **Form Size Policy / Generate pack() / Generate Resize code**.
- Click the Source tab above the design window to open the program code. Locate the main method. Use the + icon to open the program lines and change the parameter "**Nimbus**" to "**Windows**".

Run the program and accept the **main** class which is offered. Check that a blank window appears and has the correct size and colour scheme. Close the program and return to the NetBeans editing screen. Click the **Design** tab to move to the form layout view.

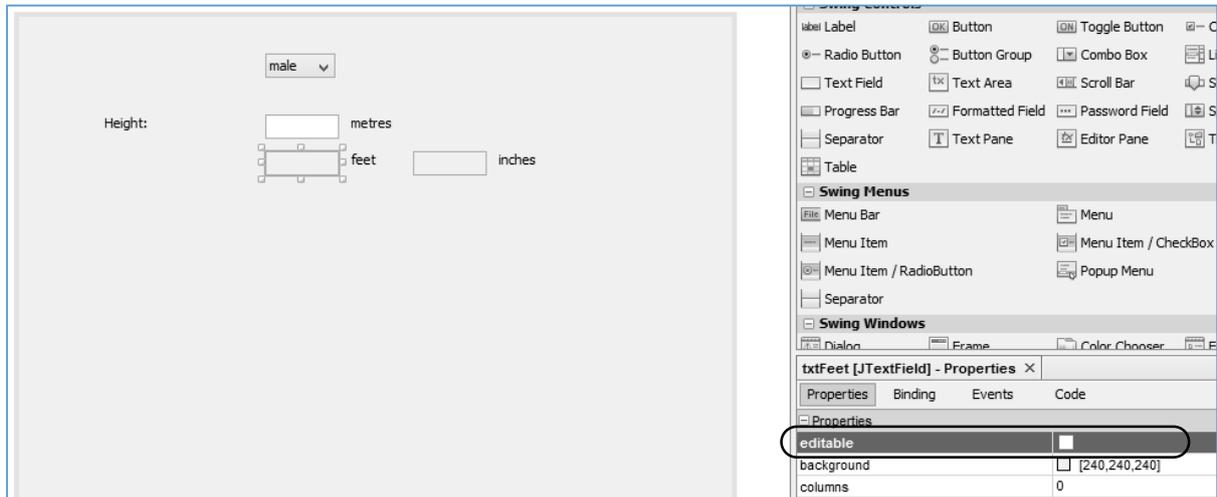
Select the **Combo Box** component from the palette and drag and drop this on the form. Rename the component as **cmbGender**. Go to the **model** property, and set the values as:

male, female

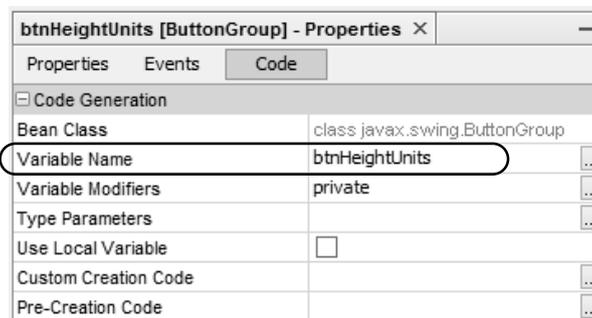


Add labels and text fields for entry of the person's height. Rename the text fields as **txtMetres**, **txtFeet** and **txtInches**.

We will allow the user to select either **metric** or **imperial** units for entry of the height. Let us first assume that metric units will be used, so remove the **tick** from the **editable** property of the **feet** and **inches** text fields.



Select the **Button Group** component from the palette and drag and drop this on the form. This is an invisible component, so nothing appears on the form. Go to the **Properties** window and select the **Code** tab, then reset the **Variable Name** to **btnHeightUnits**:



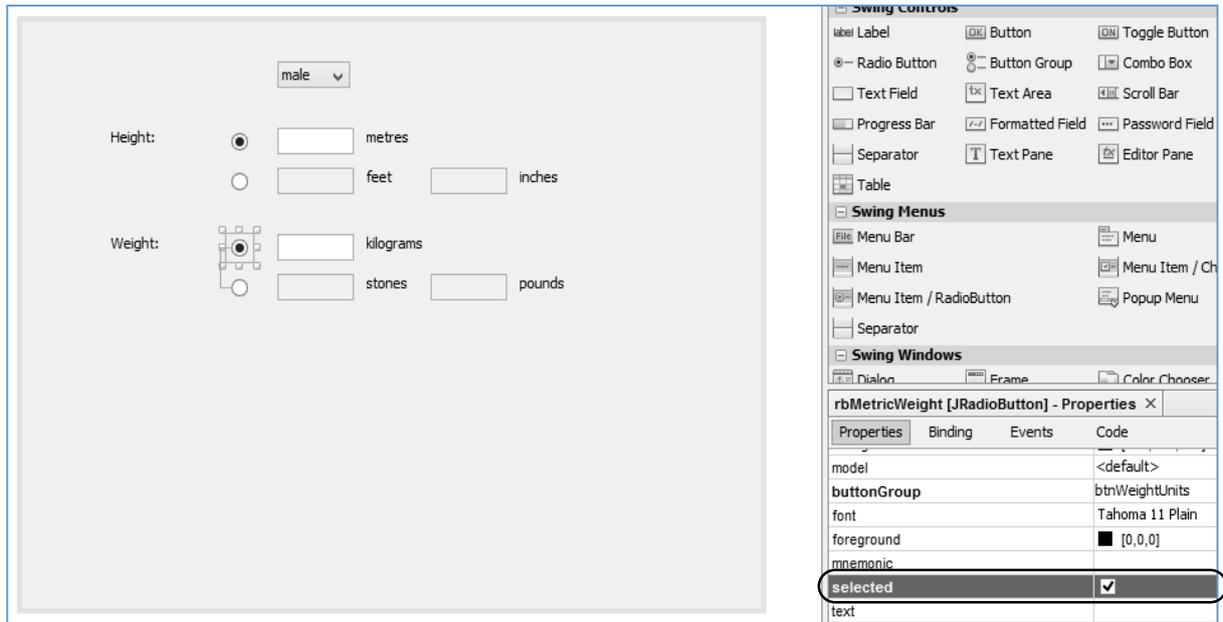
Add two **radio buttons** to the form and rename these as **rbMetricHeight** and **rbImperialHeight**. Set the **buttonGroup** property for each of these buttons to **btnHeightUnits**. This will ensure that the buttons operate together so only one of the options can be selected.

Add a tick to the **selected** property for the **rbMetricHeight** button:

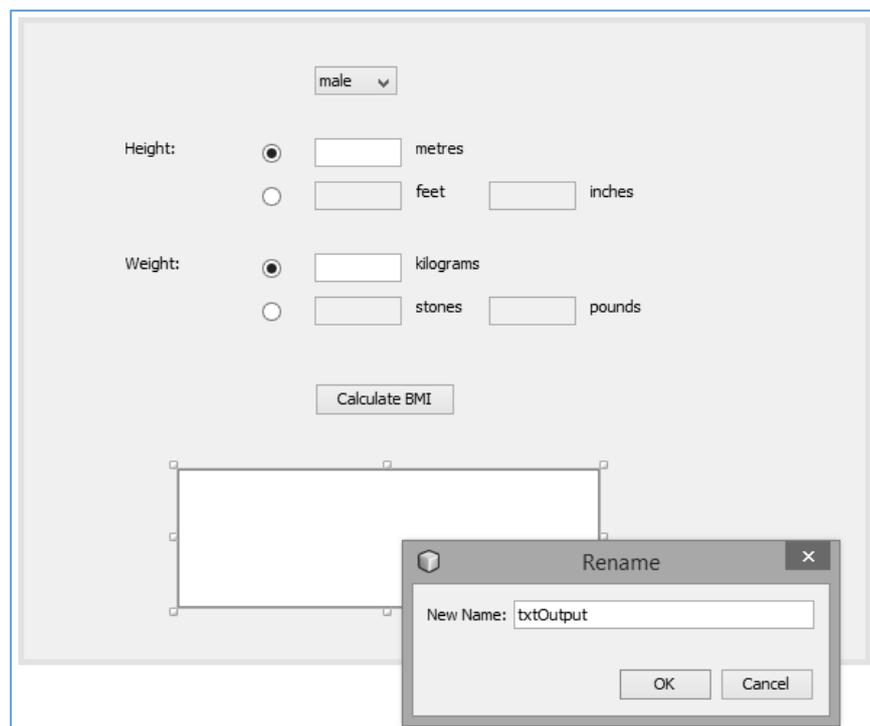


Create a set of components in a similar way for the entry of the person's weight:

- Add **labels** and **text fields**. Name the text fields as *txtKilos*, *txtStones* and *txtPounds*. Remove the **ticks** from the **editable** property of the **stones** and **pounds** text fields.
- Add a **buttonGroup** component, and rename this as *btnWeightUnits*.
- Add two **radio buttons** and rename these as *rbMetricWeight* and *rbImperialWeight*. Set the **buttonGroup** property for each of these buttons to *btnWeightUnits*. Add a tick to the **selected** property for the *rbMetricWeight* button:

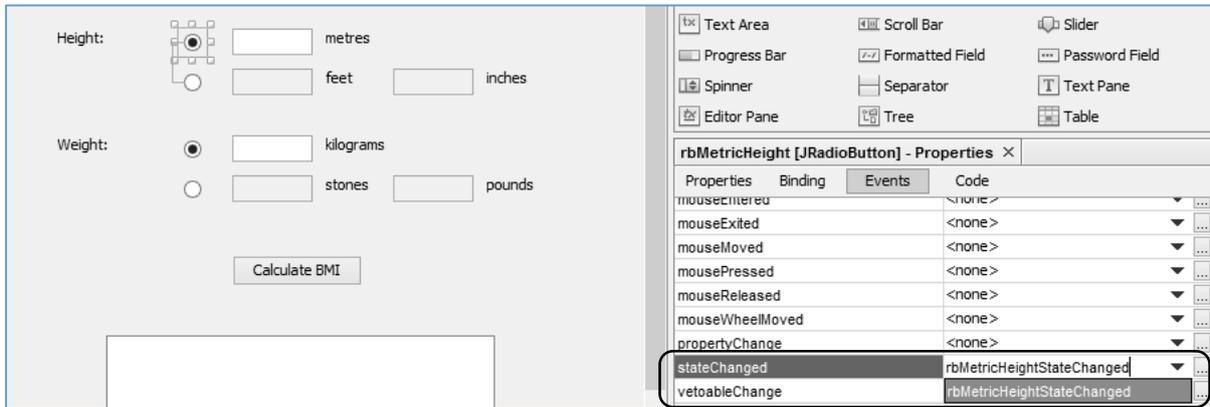


Complete the form by adding a **button** with the caption "**Calculate BMI**", and a **text area**. Rename these components as *btnCalculate* and *txtOutput*:



When the program is running, it should be possible for the user to select either metric or imperial units for entry of height and weight. When a change is made between metric and imperial units, the appropriate text fields should become editable. We will arrange this now.

Select the **rbMetricHeight** radio button. Open the **Events** tab in the **Properties** window and locate the **stateChanged** event. Select **rbMetricHeightStateChanged** from the drop down list:



Add lines of code which will detect which of the **Height** radio buttons is selected, and set the **editable** properties of the **text fields** accordingly. We will also delete any previous text entry when a text field becomes un-editable.

```
private void rbMetricHeightStateChanged(javax.swing.event.ChangeEvent evt) {
    if(rbMetricHeight.isSelected()==true)
    {
        txtFeet.setText("");
        txtInches.setText("");

        txtMetres.setEditable(true);
        txtFeet.setEditable(false);
        txtInches.setEditable(false);
    }
    else
    {
        txtMetres.setText("");

        txtMetres.setEditable(false);
        txtFeet.setEditable(true);
        txtInches.setEditable(true);
    }
}
```

Run the program. Select the radio button for **metric height**, and check that a value can be entered in the **metres** text field, whilst the feet and inches text fields are not editable. Now select the radio button for **imperial height**, and check that the **feet** and **inches** fields become editable.

Close the program and return to the NetBeans screen. Select the **Design** tab.

Select the **rbMetricWeight** radio button. Open the **Events** tab in the **Properties** window and locate the **stateChanged** event. Select **rbMetricWeightStateChanged** from the drop down list.

Add program code to the **method**:

```
private void rbMetricWeightStateChanged(javax.swing.event.ChangeEvent evt) {
    if(rbMetricWeight.isSelected()==true)
    {
        txtStones.setText("");
        txtPounds.setText("");

        txtKilos.setEditable(true);
        txtStones.setEditable(false);
        txtPounds.setEditable(false);
    }
    else
    {
        txtKilos.setText("");

        txtKilos.setEditable(false);
        txtStones.setEditable(true);
        txtPounds.setEditable(true);
    }
}
}
```

Run the program again. Select the radio button for **metric weight**, and check that a value can be entered in the **kilograms** text field, whilst the stones and pounds text fields are not editable. Now select the radio button for **imperial weight**, and check that the **stones** and **pounds** fields become editable.

Close the program and return to the NetBeans screen. Select the **Design** tab to display the form. We can now begin work on calculation of the Body Mass Index.

Double click the **"Calculate BMI"** button to create a **method**.

Add definitions for **local variables** which will be needed in the calculation, and for output of the results:

```
private void btnCalculateActionPerformed(java.awt.event.ActionEvent evt) {
    double height, weight;
    int inches, feet, stones, pounds;
    int totalInches, totalPounds;
    String s="";
    double BMI;
    String advice;
}
}
```

We will add lines of program code to input the person's height and display this in the text area in the output section of the form. We must consider two situations:

- If the height is entered in metric units, this can be output directly as **metres**.
- If the user chooses to input the height as **feet** and **inches**, this must be converted to metres. We will firstly calculate the total number of inches, then use a conversion factor:

$$1 \text{ inch} = 0.0254 \text{ metres}$$

```
private void btnCalculateActionPerformed(java.awt.event.ActionEvent evt) {
    double height, weight;
    int inches, feet, stones, pounds;
    int totalInches, totalPounds;
    String s="";
    double BMI;
    String advice;

    if (rbMetricHeight.isSelected()==true)
    {
        height=Double.parseDouble(txtMetres.getText());
    }
    else
    {
        inches=Integer.parseInt(txtInches.getText());
        feet=Integer.parseInt(txtFeet.getText());
        totalInches=feet*12 + inches;
        height=totalInches* 0.0254;
    }
    s += "Height: "+String.format("%.2f", height)+" metres \n";
    txtOutput.setText(s);
}
```

Run the program and check that the output of heights is handled correctly:

The screenshot shows a Java Swing window with a BMI calculation form. At the top, there is a dropdown menu for gender, currently set to "male". Below this, there are two sections: "Height:" and "Weight:".

In the "Height:" section, there are three radio buttons: "metres", "feet", and "inches". The "feet" radio button is selected. There are input fields for "feet" (containing the number 5) and "inches" (containing the number 10).

In the "Weight:" section, there are three radio buttons: "kilograms", "stones", and "pounds". The "kilograms" radio button is selected. There are empty input fields for "kilograms", "stones", and "pounds".

Below the form is a button labeled "Calculate BMI".

At the bottom of the window, there is a text area containing the output: "Height: 1.78 metres".

Close the program and return to the editing screen. Add a further section of code to the **btnCalculateActionPerformed** method to enter and display **weights**, converting from **stones** and **pounds** to **kilograms** if necessary:

```

    if (rbMetricHeight.isSelected()==true)
    {
        height=Double.parseDouble(txtMetres.getText());
    }
    else
    {
        inches=Integer.parseInt(txtInches.getText());
        feet=Integer.parseInt(txtFeet.getText());
        totalInches=feet*12 + inches;
        height=totalInches* 0.0254;
    }
    s += "Height: "+String.format("%.2f", height)+" metres \n";

    if (rbMetricWeight.isSelected()==true)
    {
        weight=Double.parseDouble(txtKilos.getText());
    }
    else
    {
        stones=Integer.parseInt(txtStones.getText());
        pounds=Integer.parseInt(txtPounds.getText());
        totalPounds=stones*14 + pounds;
        weight=totalPounds* 0.454;
    }
    s += "Weight: "+String.format("%.1f", weight)+" kilograms \n";

    txtOutput.setText(s);
}

```

Run the program and check that the output of weights is correct:

Close the program and return to the editing screen.

We now have the height and weight data in the correct metric units for calculation of the Body Mass Index. Add the formula to do this:

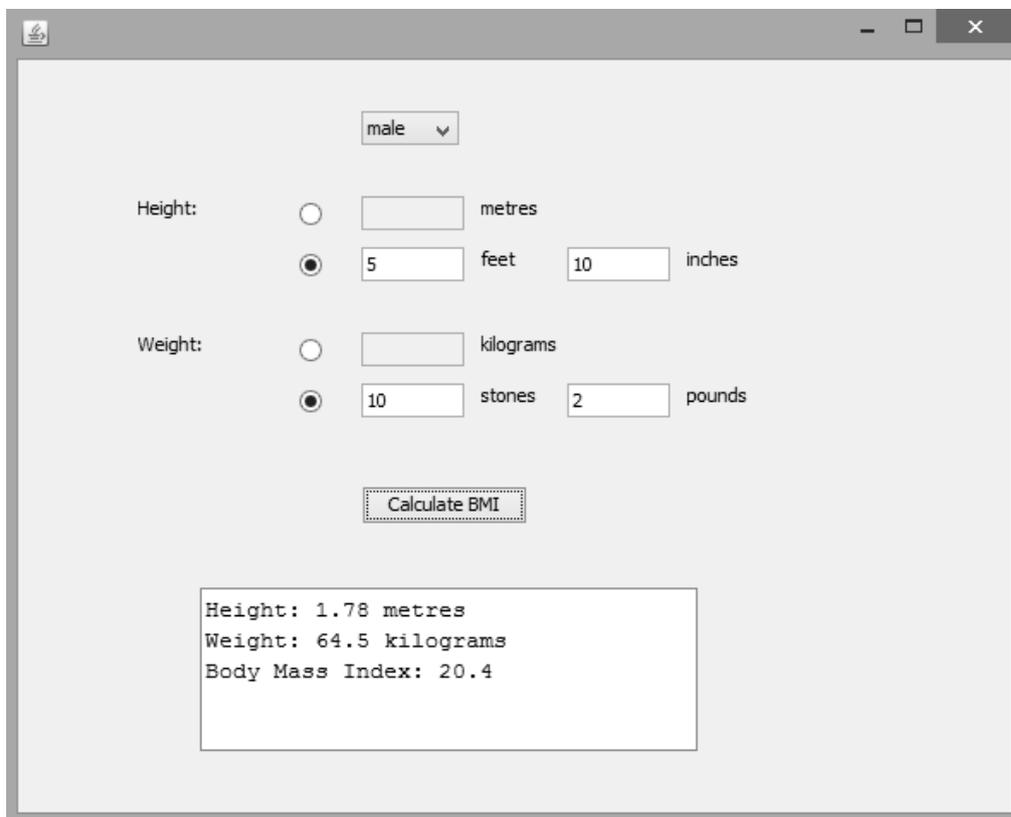
```
if (rbMetricWeight.isSelected()==true)
{
    weight=Double.parseDouble(txtKilos.getText());
}
else
{
    stones=Integer.parseInt(txtStones.getText());
    pounds=Integer.parseInt(txtPounds.getText());
    totalPounds=stones*14 + pounds;
    weight=totalPounds* 0.454;
}
s += "Weight: "+String.format("%.1f", weight)+" kilograms \n";

BMI = weight/(height*height);

s += "Body Mass Index: "+String.format("%.1f", BMI)+" \n";

txtOutput.setText(s);
}
```

Run the program and test the BMI calculation. To verify that the program is working correctly, carry out the calculations a second time using a calculator or spreadsheet:



The screenshot shows a Java Swing application window with a light gray background. At the top, there is a dropdown menu with 'male' selected. Below this, there are two sections for input: 'Height' and 'Weight'. The 'Height' section has a radio button for 'metres' (unselected) and a radio button for 'feet' (selected). The 'feet' option has two input fields: '5' for feet and '10' for inches. The 'Weight' section has a radio button for 'kilograms' (unselected) and a radio button for 'stones' (selected). The 'stones' option has two input fields: '10' for stones and '2' for pounds. Below the input fields is a button labeled 'Calculate BMI'. At the bottom of the window is a text area containing the following output:

```
Height: 1.78 metres
Weight: 64.5 kilograms
Body Mass Index: 20.4
```

Close the program and return to the editing screen.

The final task is to link **Body Mass Index** and **gender**, to give advice on whether the person is overweight, in the healthy range, or underweight. This can be done by checking whether “male” or “female” is selected in the **cmbGender** combo box, then using a series of **IF...THEN...ELSE** commands to check BMI value against the specified ranges for overweight, healthy or underweight:

```
BMI = weight/(height*height);

s += "Body Mass Index: "+String.format("%.1f", BMI)+" \n";

if (cmbGender.getSelectedItem().equals("male"))
{
    if (BMI > 25)
    {
        advice="Overweight";
    }
    else
    {
        if (BMI >=20)
        {
            advice ="Healthy weight";
        }
        else
        {
            advice ="Underweight";
        }
    }
}
else
{
    if (BMI > 24)
    {
        advice="Overweight";
    }
    else
    {
        if (BMI >=19)
        {
            advice ="Healthy weight";
        }
        else
        {
            advice ="Underweight";
        }
    }
}

s += advice;

txtOutput.setText(s);
}
```

This completes the project. Run the program and check that correct advice is given for men and women with a range of different heights and weights:

A Java Swing window titled "Calculate BMI" for a male user. The gender is set to "male" in a dropdown menu. The height is entered as 6 feet and 1 inch. The weight is entered as 13 stones and 9 pounds. The "Calculate BMI" button is highlighted. The output box displays the following information:

```
Height: 1.85 metres
Weight: 86.7 kilograms
Body Mass Index: 25.2
Overweight
```

A Java Swing window titled "Calculate BMI" for a female user. The gender is set to "female" in a dropdown menu. The height is entered as 5 feet and 7 inches. The weight is entered as 10 stones and 2 pounds. The "Calculate BMI" button is highlighted. The output box displays the following information:

```
Height: 1.70 metres
Weight: 64.5 kilograms
Body Mass Index: 22.3
Healthy weight
```