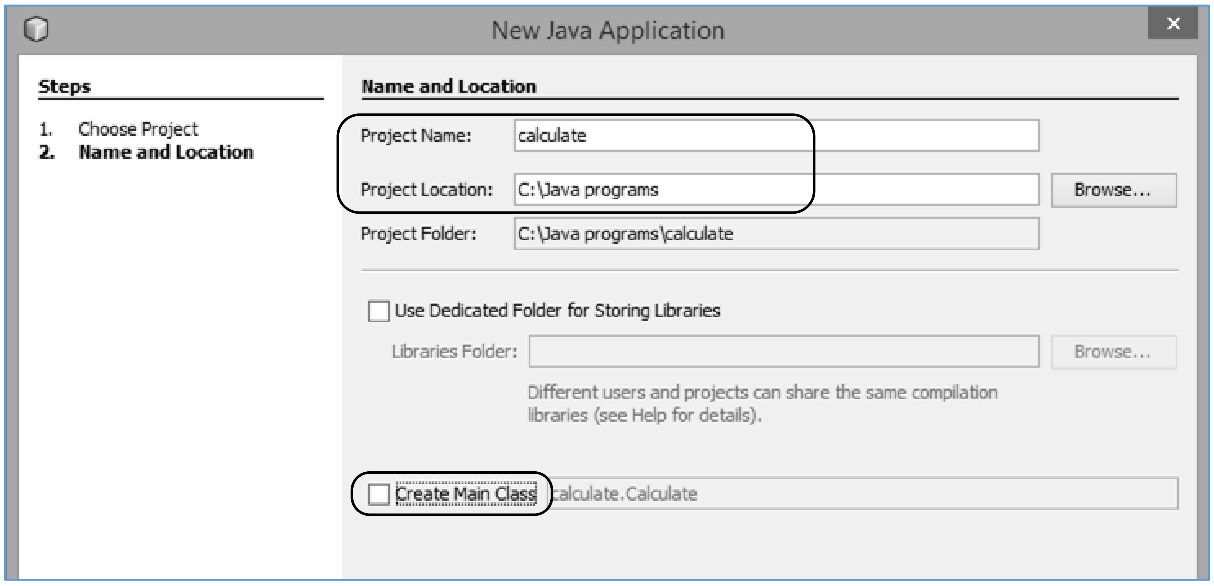
2 Calculations

Most computer programs need to carry out calculations, for example: with money, quantities of materials, or dates and times. In this chapter, we will examine how the Java programming language handles numbers.

Our first program is a simple calculator, which will carry out the arithmetic operations of adding, subtracting, multiplying and dividing numbers.

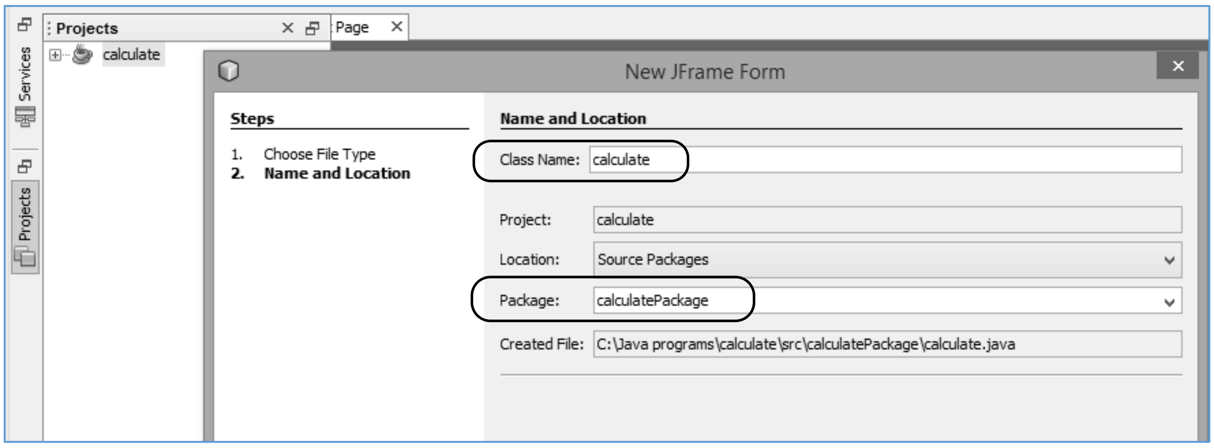
Go to the ***File*** menu in NetBeans and ***Close all projects***. Select the ***New Project*** option and check that ***Java*** / ***Java application*** is highlighted, then click the ***Next*** button. Give the ***Project Name*** as ***calculate***, and un-tick the ***Create Main Class*** option:



Project Name: calculate
Project Location: C:\Java programs
Project Folder: C:\Java programs\calculate
☐ Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).
☐ Create Main Class calculate.Calculate

Click the ***Finish*** button to return to the NetBeans main page.

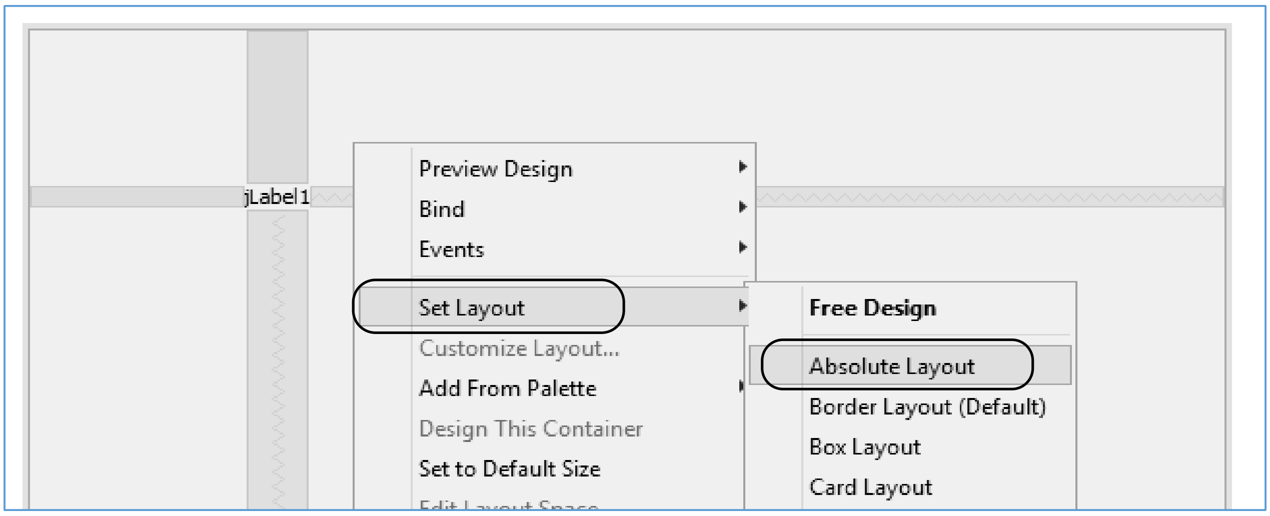
Click the ***Projects*** tab at the left of the page, then right-click on the ***calculate*** project icon to open a drop-down menu. Select ***New***, then ***JFrame Form***. Set the ***Class Name*** as ***calculate***, and the ***Package*** as ***calculatePackage***:



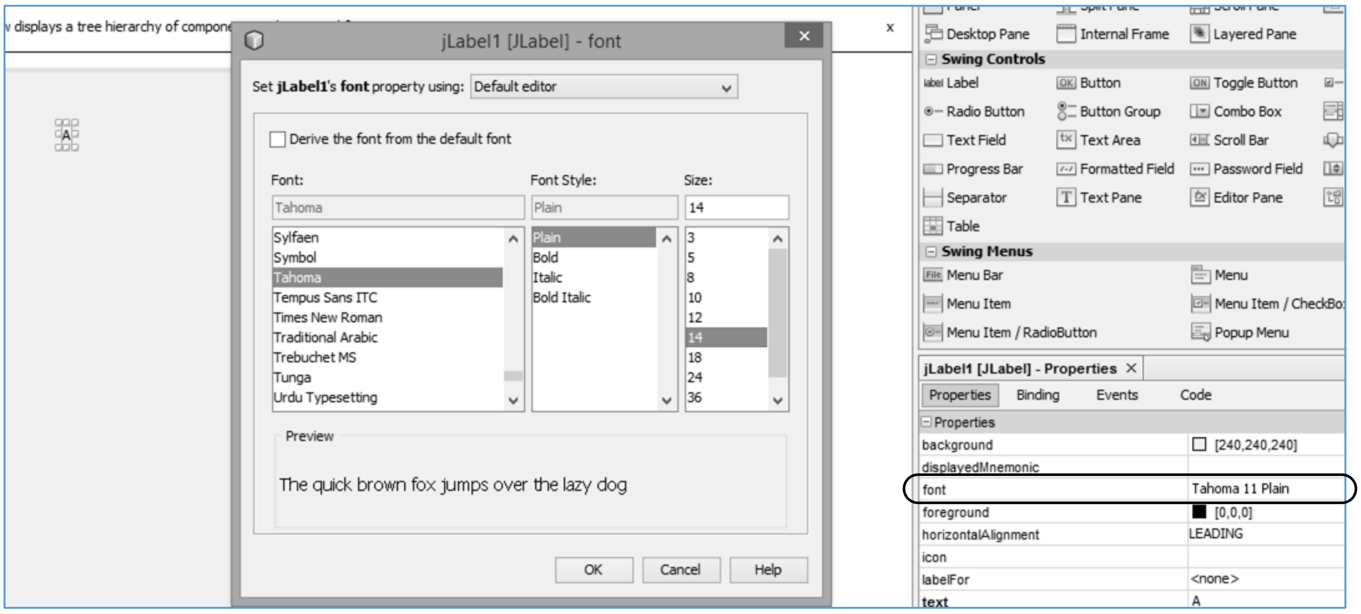
Class Name: calculate
Project: calculate
Location: Source Packages
Package: calculatePackage
Created File: C:\Java programs\calculate\src\calculatePackage\calculate.java

Return to the NetBeans design screen. Locate the ***Label*** component in the ***Palette***, and drag and drop a label onto the form. Notice that grey bars are shown linked to the label component which are not required.

Move to an empty area of the ***form*** and right-click to show the drop-down menu. Select ***Set Layout*** / ***Absolute Layout***:



Right-click on the label which you placed on the form, and use the Edit Text option to change the caption to the letter '***A***':



Set **jLabel1**'s **font** property using: Default editor

☐ Derive the font from the default font

Font:	Font Style:	Size:
Tahoma	Plain	14

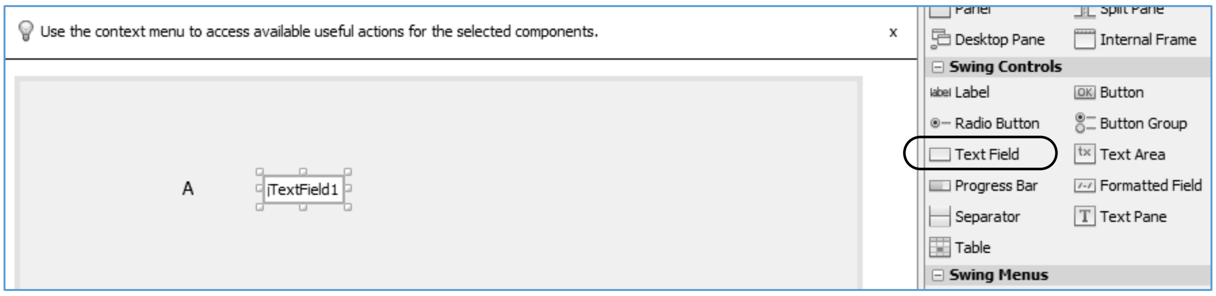
Preview

The quick brown fox jumps over the lazy dog

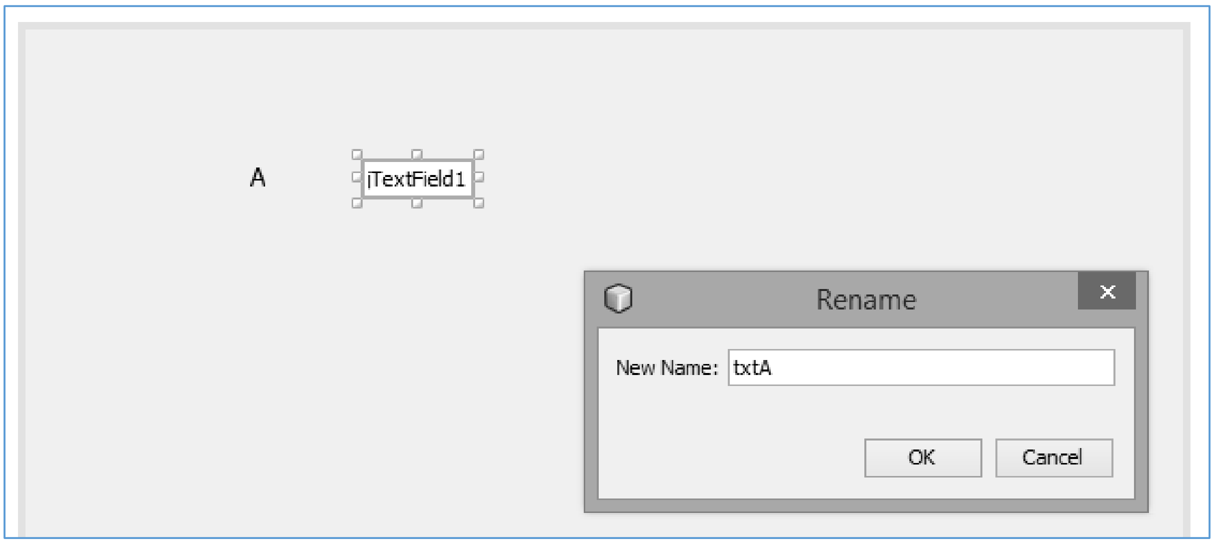
Properties	
background	☐ [240,240,240]
displayedMnemonic	
font	Tahoma 11 Plain
foreground	■ [0,0,0]
horizontalAlignment	LEADING
icon	
labelFor	<none>
text	A

Increase the size of the label text by clicking on the ***font*** property in the ***Properties*** window, then resetting the ***font size*** to 14.

Locate the ***Text Field*** component in the ***Palette***. Drag and drop a ***text field*** next to the label:

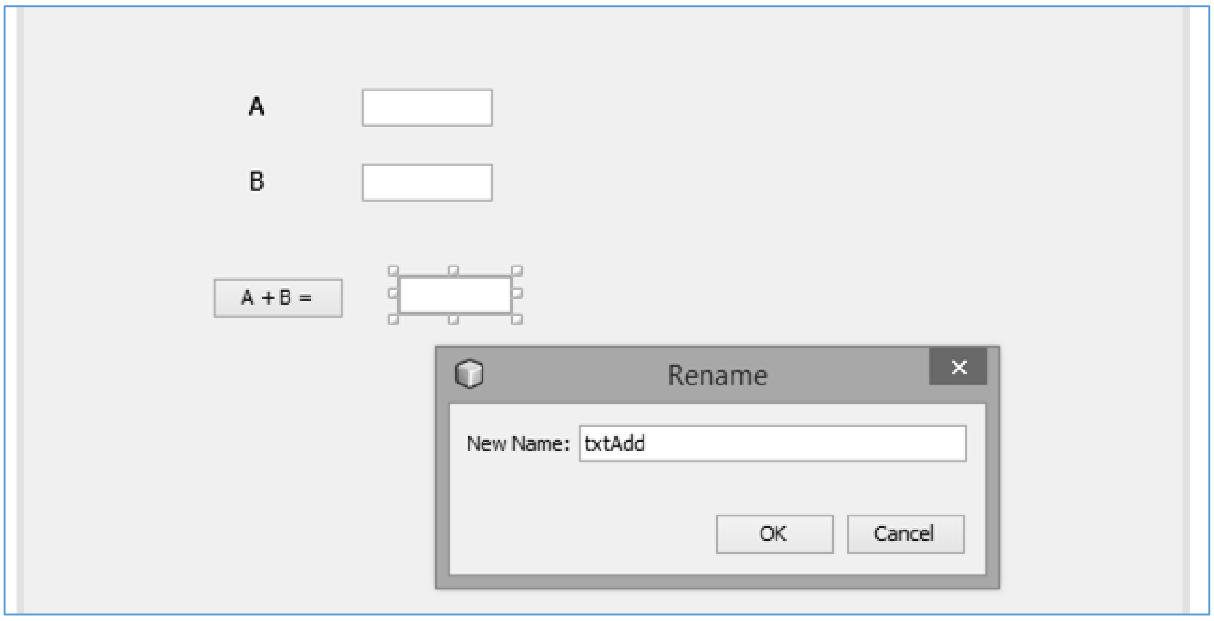


Right-click on the text field. Select the ***Change Variable Name*** option from the drop-down menu, and give the name '***txtA***'. Right-click again and select ***Edit Text***. Delete the caption from the text field.



Add another ***label*** and ***text field***, naming the text field as '**txtB**' and deleting the text caption.

Finally add a button with the caption '***A + B =***', and a text field with the name '***txtAdd***'. Delete the caption from the text field



When the program runs, we would like to enter numbers in the text fields labelled A and B, click the button to add the numbers together, then display the answer in the ***txtAdd*** text field.

The first stage is to set up ***variables***. We can think of these as memory boxes which will store the two numbers entered, and the answer when it is calculated. We will set the ***data type*** as ***integer***, which means 'a whole number'.

Click the ***Source*** tab to display the program code. Near the top of the program, just below the start of the ***calculate*** class, add the line of code shown below:

```
package calculatePackage;

public class calculate extends javax.swing.JFrame {

    int A, B, answer;

    public calculate() {

        initComponents();
    }
```

Click the ***Design*** tab to return to the design screen, then double click the '***A + B =***' button. The ***Code*** page will open and you will see that an empty button click ***method*** has been created. Add lines of code as shown:

```
private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {

    A=Integer.parseInt(txtA.getText());
    B=Integer.parseInt(txtB.getText());
    answer=A+B;
    txtAdd.setText(Integer.toString(answer));
 }
```

Quite a lot is going on here! The line:

A = Integer.parseInt(txtA.getText( ));

is telling the program to collect a text entry from the text field ***txtA***, then convert this to an integer number and store it as the variable ***A***. The number ***B*** is collected in a similar way.

The line:

answer = A+B;

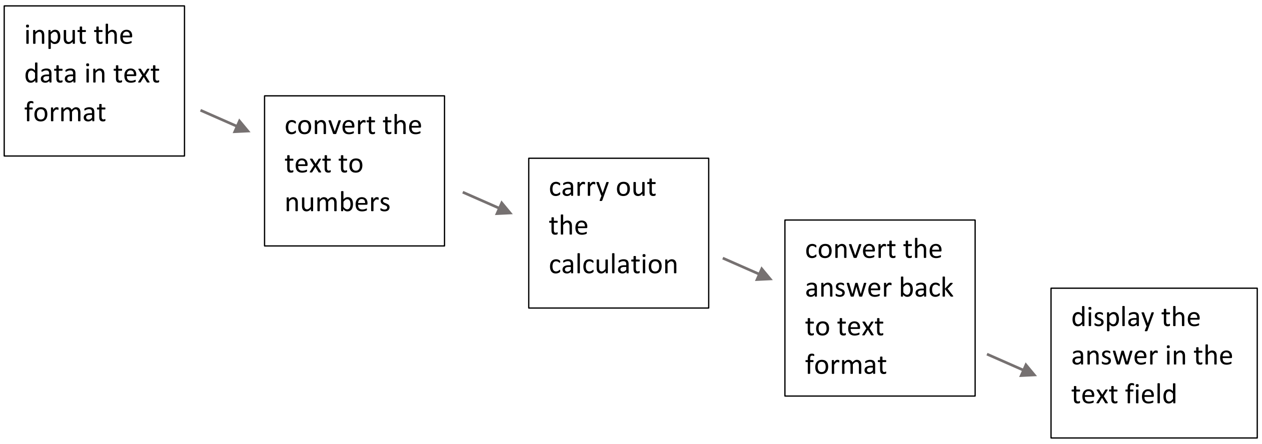
adds the numbers together and stores the result as the variable called ***answer***.

Finally, the answer is displayed in the text field ***txtAdd*** by the line:

txtAdd.setText(Integer.toString(answer));

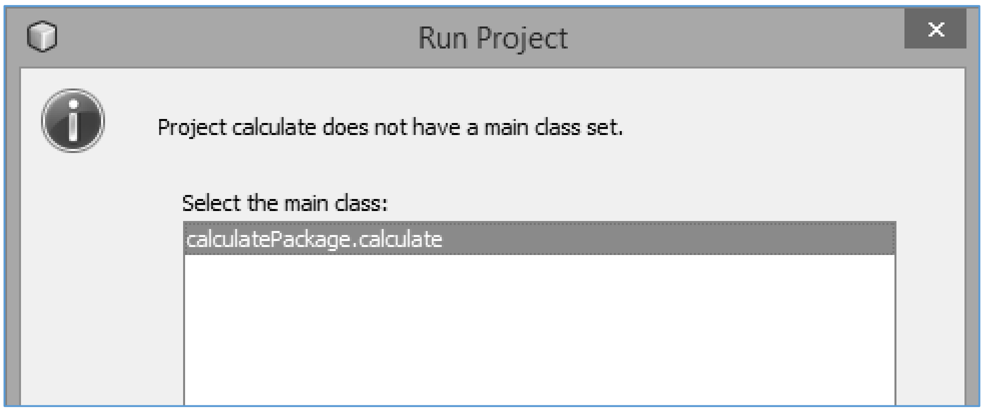
The term '***string***' refers to data in text format. The integer number must be converted to text before it can be displayed in a text field.

The full sequence can be represented as:


input the data in text format
convert the text to numbers
carry out the calculation
convert the answer back to text format
display the answer in the text field

We are now ready to test the program.

Click the green ***Run*** button, and accept ***calculatePackage.calculate*** as the main class:

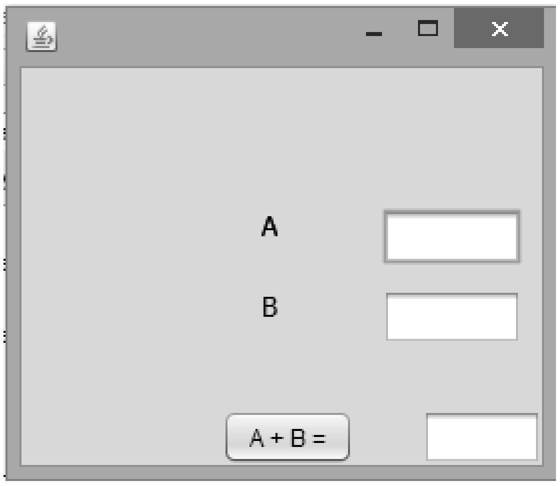


Project calculate does not have a main class set.

Select the main class:

calculatePackage.calculate

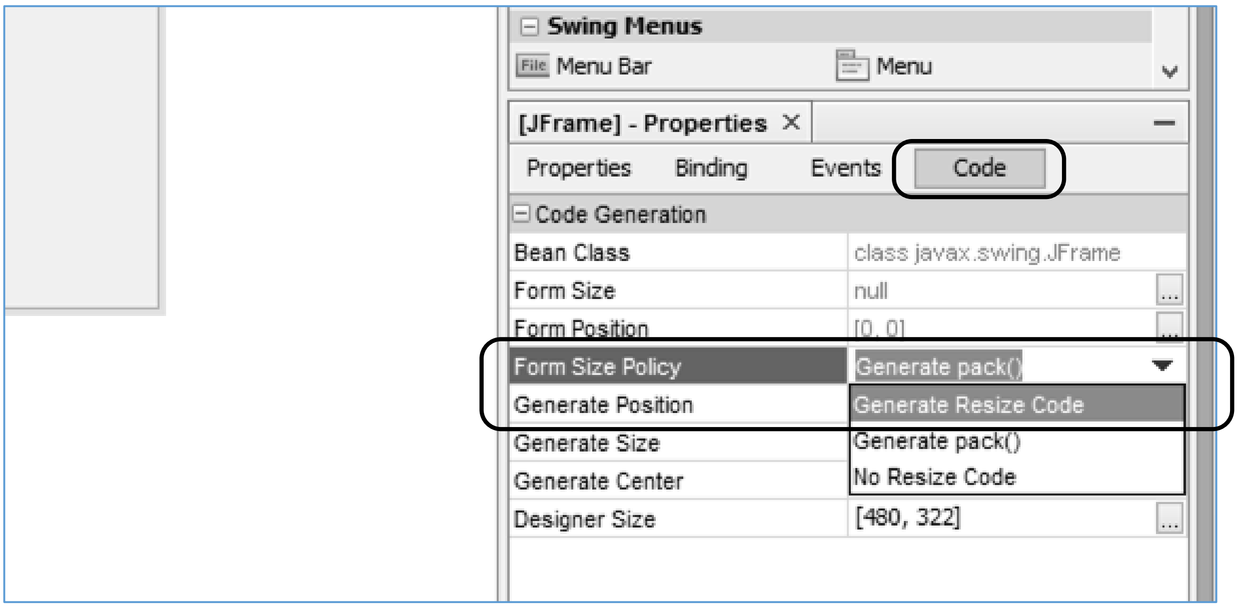
The form displays the labels, text fields and button. However, the normal Windows colour scheme is not used and the window size has not been set correctly:



Close the program and return to the NetBeans editing screen. Select the ***Source*** tab to move to the program code window, then reset "***Nimbus***" to "***Windows***" in the ***main*** method:

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

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

             if ("Windows".equals(info.getName())) {

                 javax.swing.UIManager.setLookAndFeel(info.getClassName());
                 break;
             }
         }
     } catch (ClassNotFoundException ex) {
```

Return to the ***Design*** page. Click on the ***form***, then go to the ***Code*** section of the ***Properties*** window. Select the ***Form Size Policy*** / ***Generate Resize Code*** option:

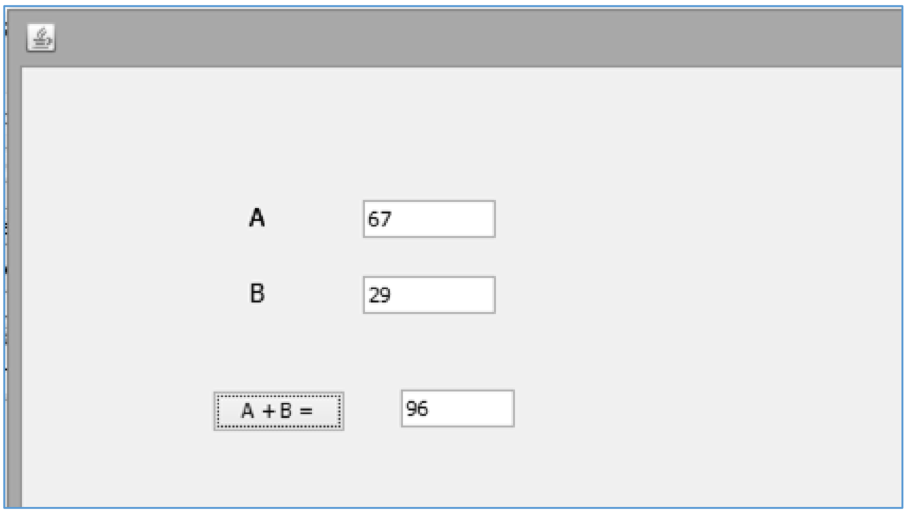


Code Generation

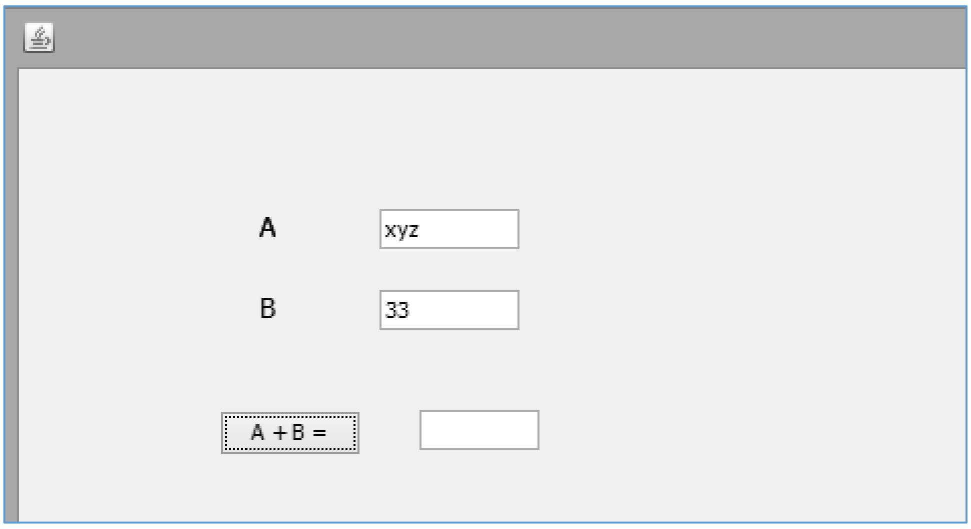
Bean Class	class javax.swing.JFrame
Form Size	null
Form Position	[0, 0]
Form Size Policy	Generate pack()
Generate Position	Generate Resize Code
Generate Size	Generate pack()
Generate Center	No Resize Code
Designer Size	[480, 322]

Re-run the program. The standard ***Windows*** colour scheme will now be used, and window size should be correct.

Test the addition function by entering two whole numbers, then clicking the button:



When software is in use, it is not unusual for mistakes to be made when entering data. For example, the user might accidentally enter a letter instead of a number. Let's investigate what would happen in this case. Enter some letters, then click the button:



You will find that nothing happens in the program window, although error messages appear in a window at the bottom of the NetBeans editing page. The program knows that the data entered should be an integer number, and anything else is ignored. However, it would be better to give a warning in the program that an error has occurred - we can do this by displaying a pop-up message box on the screen.

Close the program window and click the ***Source*** tab to show the program code. Add a line near the top of the program to load some extra code needed to create the message box:

```
package calculatePackage;

import javax.swing.JOptionPane;

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

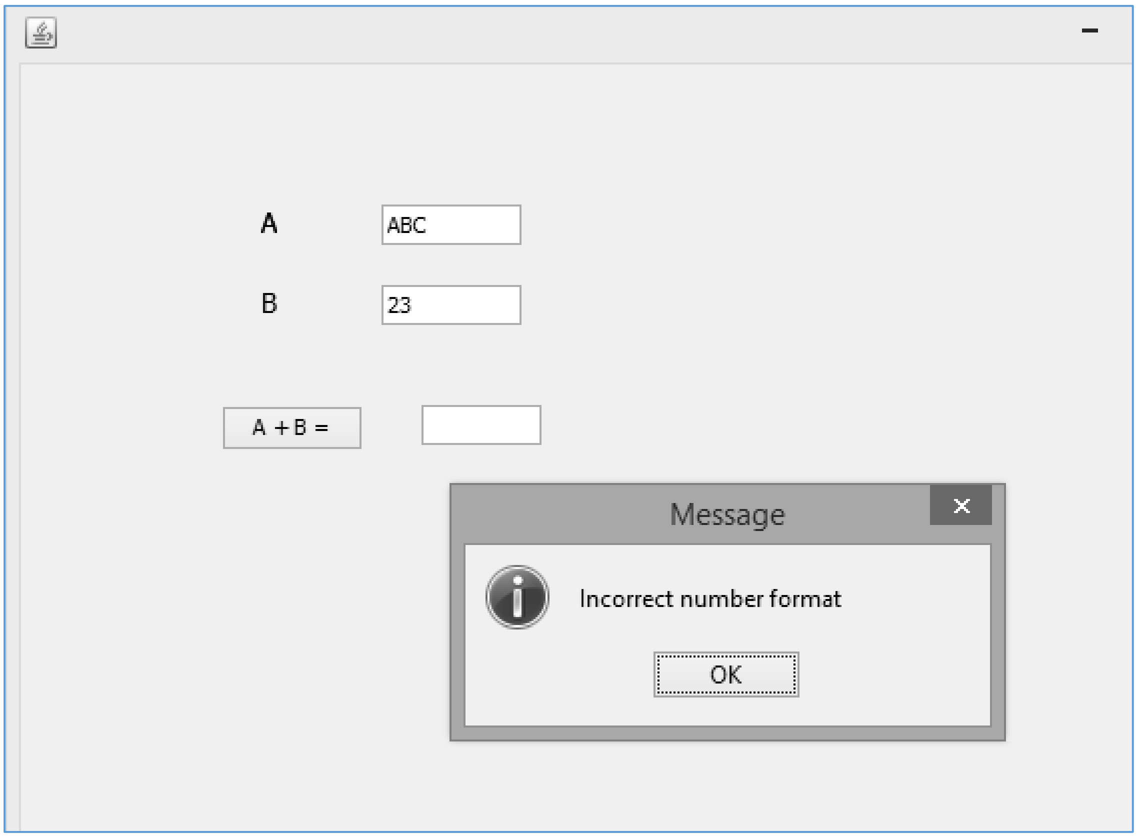
Scroll down the program listing to find the ***btnAddActionPerformed*** method which you produced earlier. We can now add a ***try...catch*** structure, as shown below:

- When the program runs, the computer will attempt to carry out the calculation.
- If an error is found in the number format, the ***catch*** block will operate. This will display a pop-up message box to warn the user.

Add these lines of code to the method, then run the program.

```
private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {
        A=Integer.parseInt(txtA.getText());
        B=Integer.parseInt(txtB.getText());
        answer=A+B;
        txtAdd.setText(Integer.toString(answer));
    }
    catch(NumberFormatException e)
    {
        JOptionPane.showMessageDialog(calculate.this, "Incorrect number format");
    }
}
```

If letters are now entered instead of numbers, the message box should appear:



Incorrect number format

We can now extend our simple calculation program to carry out the other arithmetic operations: subtracting, multiplying and dividing.

The program we have written so far uses ***integers***, whole numbers, to store the input values and the answer. This could lead to problems when we divide numbers, as the answer is unlikely to be a whole number – for example: if 13 is divided by 5. In any case, we may wish to carry out calculations using decimal fractions such as: 3.142. Fortunately, there is a simple solution – we can change the program to use the ***double*** data type which handles decimal fractions.

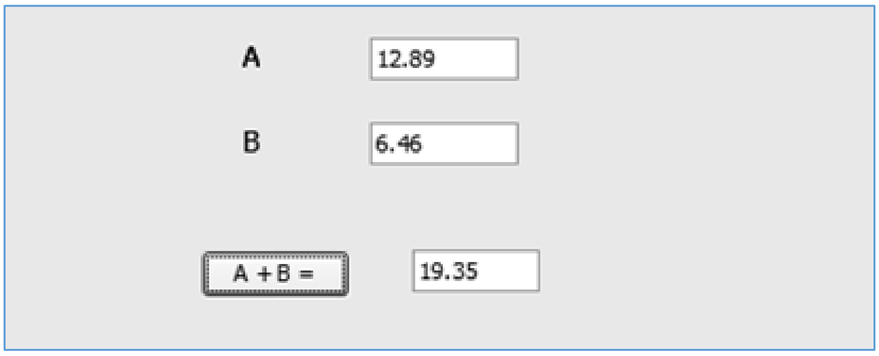
Close the program window and click the ***Source*** tab to go to the program code page. Change the variable type to ***double*** at the start of the program.

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

    double A, B, answer;

    public calculate() {
        initComponents();
    }
```

We also need to change ***Integer*** to ***Double*** each time it occurs in the ***btnAdd ActionPerformed*** method:

```
private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {
      A=Double.parseDouble(txtA.getText());
      B=Double.parseDouble(txtB.getText());

      answer=A+B;

      txtAdd.setText(Double.toString(answer));

    }
    catch(NumberFormatException e)
    {
       JOptionPane.showMessageDialog(calculate.this, "Incorrect number format");
    }
 }
```

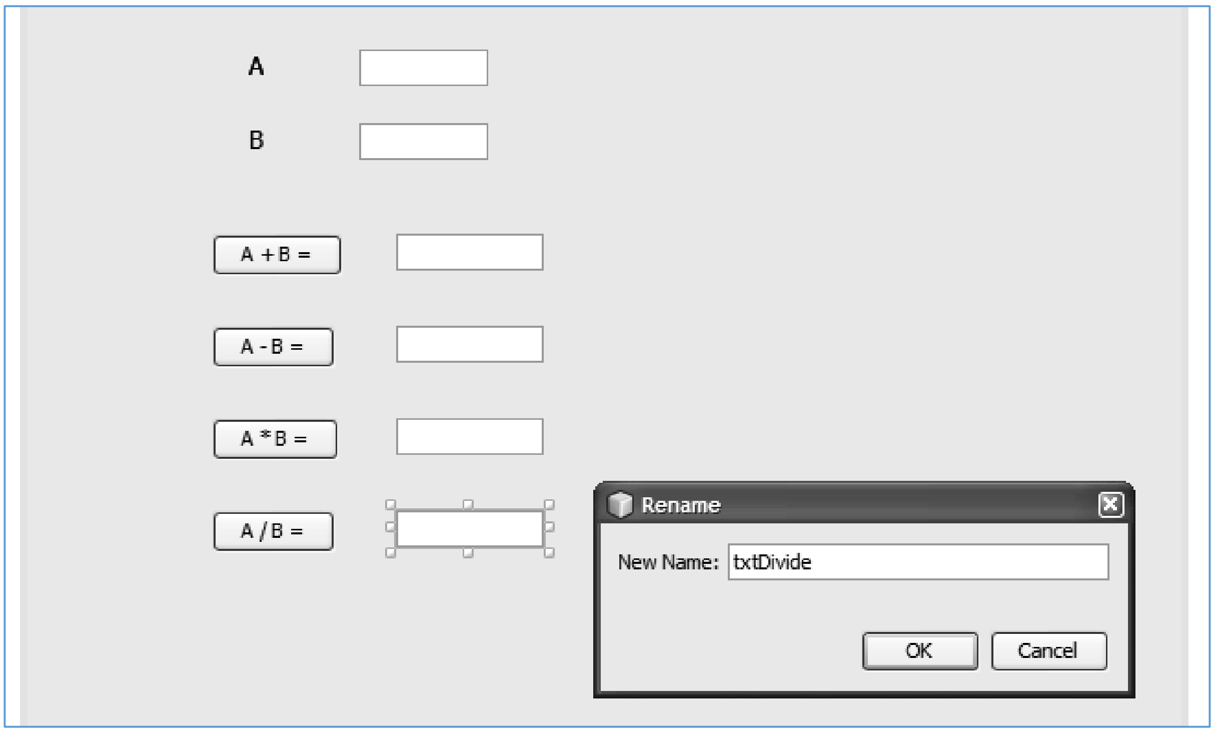
Run the program and check that addition can now be carried out with decimal numbers:



We can now set up buttons and text fields for subtracting, multiplying and dividing. Change the names of the buttons and text fields to:

btnSubtract	***txtSubtract***
btnMultiply	***txtMultiply***
btnDivide	***txtDivide***

Delete the text entries from the text fields, then drag the text field boxes to the sizes required.

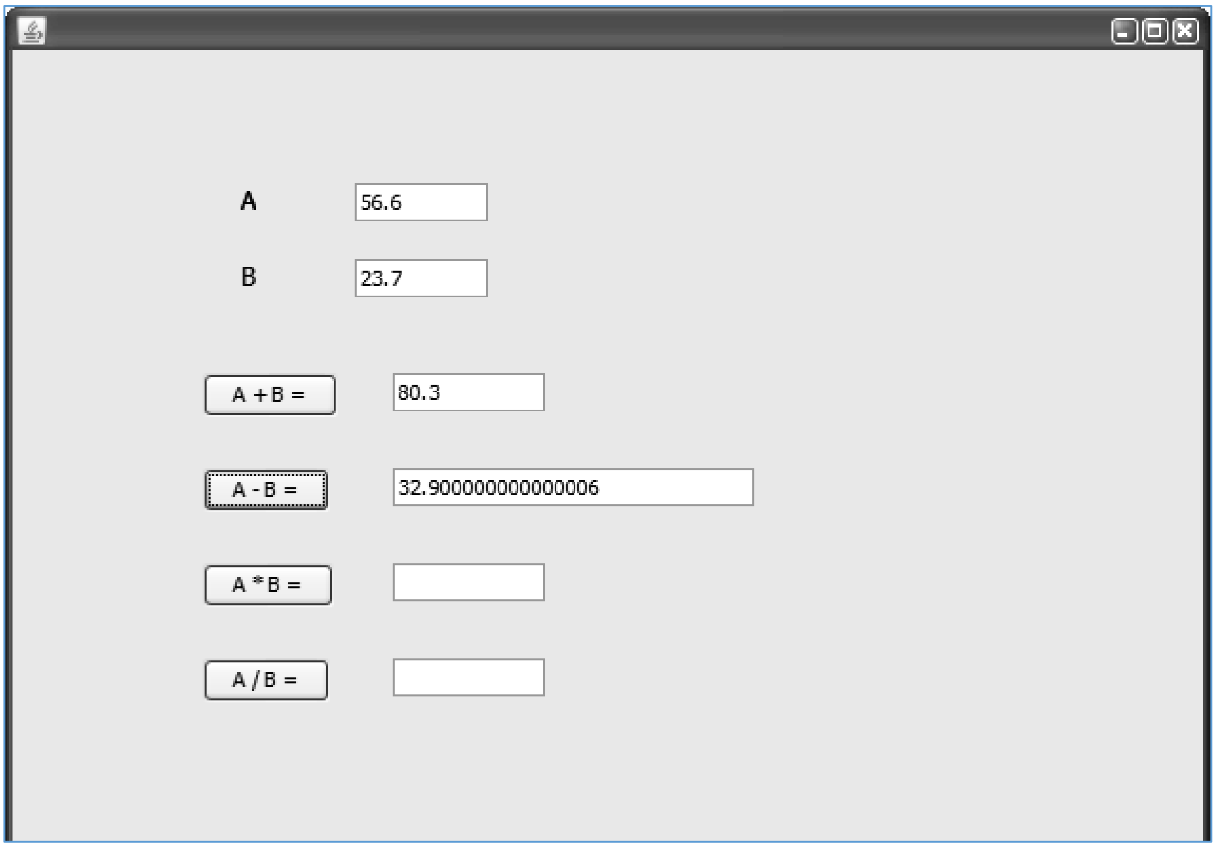


Double click the "***A – B =***" button to create an empty button click method. Copy and paste the code from the ***btnAddActionPerformed*** method, just changing the calculation and output lines as shown below:

```
private void btnSubtractActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {
       A=Double.parseDouble(txtA.getText());
       B=Double.parseDouble(txtB.getText());
       answer=A-B;
       txtSubtract.setText(Double.toString(answer));
     }
     catch(NumberFormatException e)
     {
        JOptionPane.showMessageDialog(calculate.this, "Incorrect number format");
     }
  }
```

Run the program and carry out some subtractions using decimal numbers. You will probably discover a problem: calculations using ***double*** numbers are sometimes not exactly accurate! The size of the text field has been increased to demonstrate this.

Double numbers are stored using a method called ***floating point format***. This has the advantage that very large and very small numbers can be stored in a compact way which saves memory space, so we generally accept the problem of slight inaccuracy in the final decimal place.

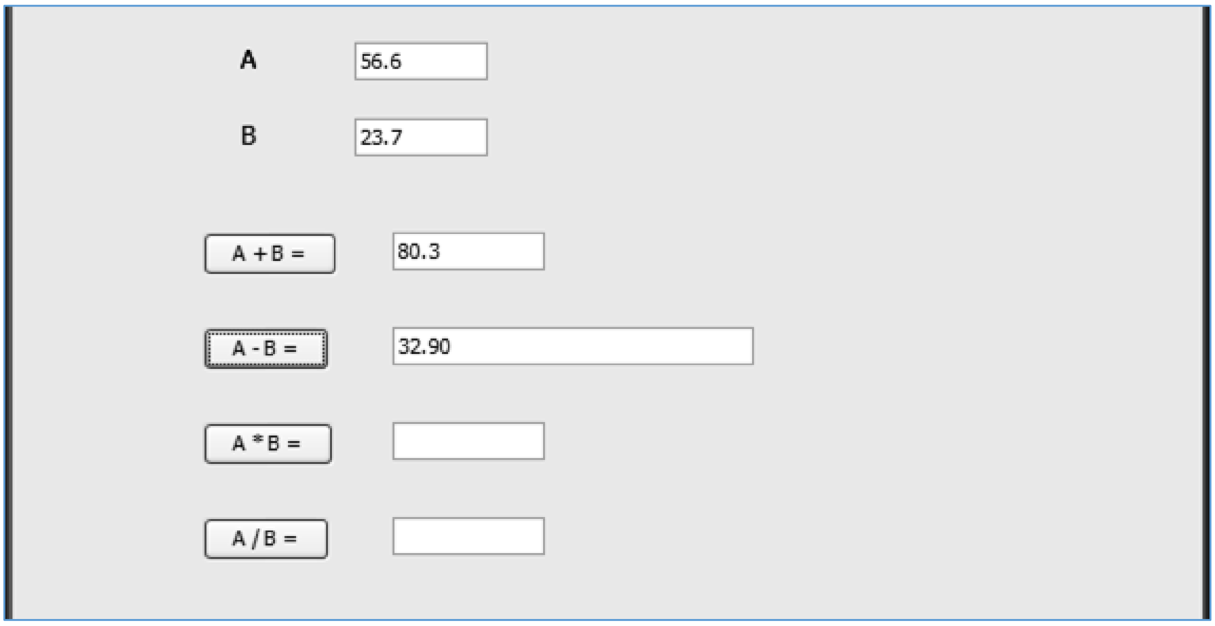


Close the program window and return to the NetBeans editing screen.

We can easily display an accurate answer by limiting the number of decimal places. Change the output line to include the formatting instruction "***%.2f***". This will produce an answer to two decimal places.

```
    private void btnSubtractActionPerformed(java.awt.event.ActionEvent evt) {
       try
       {
         A=Double.parseDouble(txtA.getText());
         B=Double.parseDouble(txtB.getText());
         answer=A-B;

         txtSubtract.setText(String.format("%.2f",answer));

       }
       catch(NumberFormatException e)
       {
          JOptionPane.showMessageDialog(calculate.this, "Incorrect number format");
       }
    }
```

Re-run the program and check that a correct answer is now displayed.



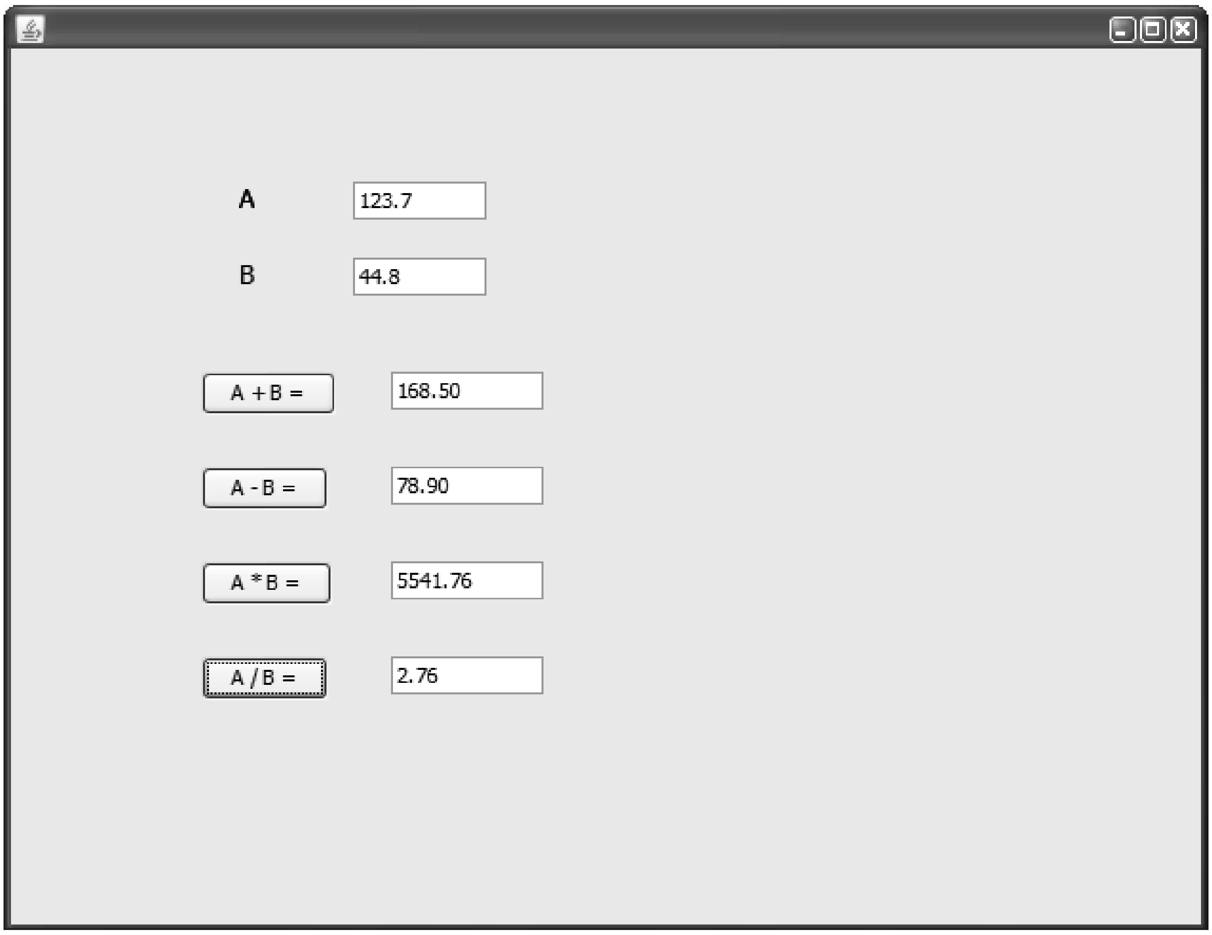
Close the program window and return to the Design screen. Edit the ***btnAdd( )*** method to format the output to two decimal places.

```
private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {
      try
      {
        A=Double.parseDouble(txtA.getText());
        B=Double.parseDouble(txtB.getText());
        answer=A+B;

        txtAdd.setText(String.format("%.2f",answer));

      }
      catch(NumberFormatException e)
```

Create button click methods for the two remaining buttons. Add program code to carry out multiplication and division. Remember to change the calculation lines to use the operators '*****' and '**/**' for multiplication and division:

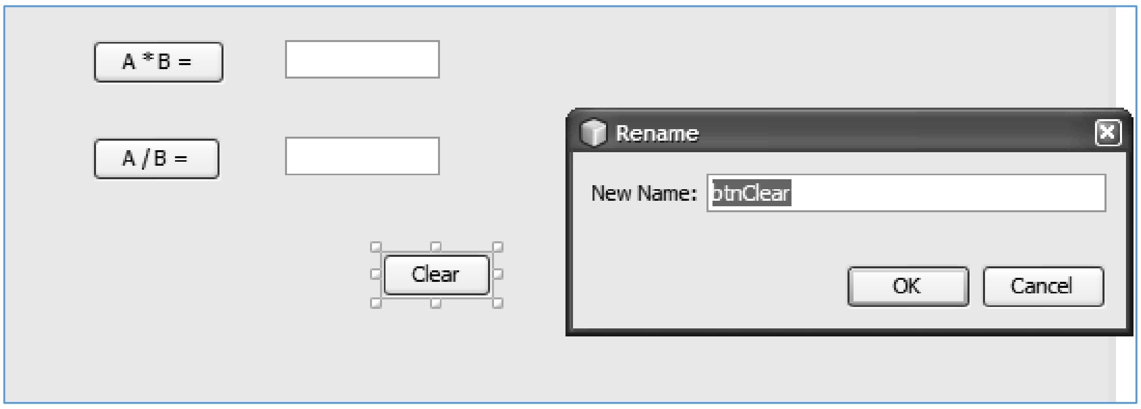
```
private void btnMultiplyActionPerformed(java.awt.event.ActionEvent evt) {
   try
   {
     A=Double.parseDouble(txtA.getText());
     B=Double.parseDouble(txtB.getText());

     answer=A*B;
     txtMultiply.setText(String.format("%.2f",answer));

   }
   catch(NumberFormatException e)
   {
      JOptionPane.showMessageDialog(calculate.this, "Incorrect number format");
   }
 }
```

Carry out tests on the program. To be certain that the program is working correctly, you should check the answers independently by means of a calculator or spreadsheet.



A: 123.7
B: 44.8
A +B = 168.50
A - B = 78.90
A * B = 5541.76
A / B = 2.76

One final improvement is to add a ***Clear*** button which will clear all the text fields, ready for the next calculation. Right-click the button to edit the caption to '***Clear***', then rename the button as ***btnClear***.



New Name: btnClear

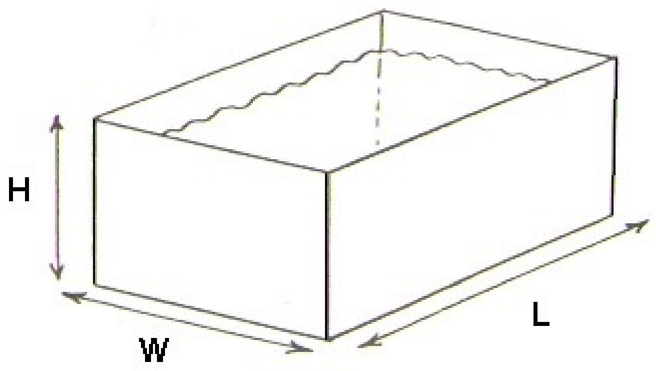
Double click the ***Clear*** button and add lines of code to the method:

```
private void btnClearActionPerformed(java.awt.event.ActionEvent evt) {

      txtA.setText("");
      txtB.setText("");
      txtAdd.setText("");
      txtSubtract.setText("");
      txtMultiply.setText("");
      txtDivide.setText("");

  }
```

Run the finished program and check that the text fields can be cleared correctly.

We have now discovered a lot about the way that Java handles calculations, and are ready to use these techniques in a more realistic application:

A company manufactures rectangular water storage tanks from sheet metal. The bottom and side panels of the tank are welded together without any overlap.



A program is required which will input:

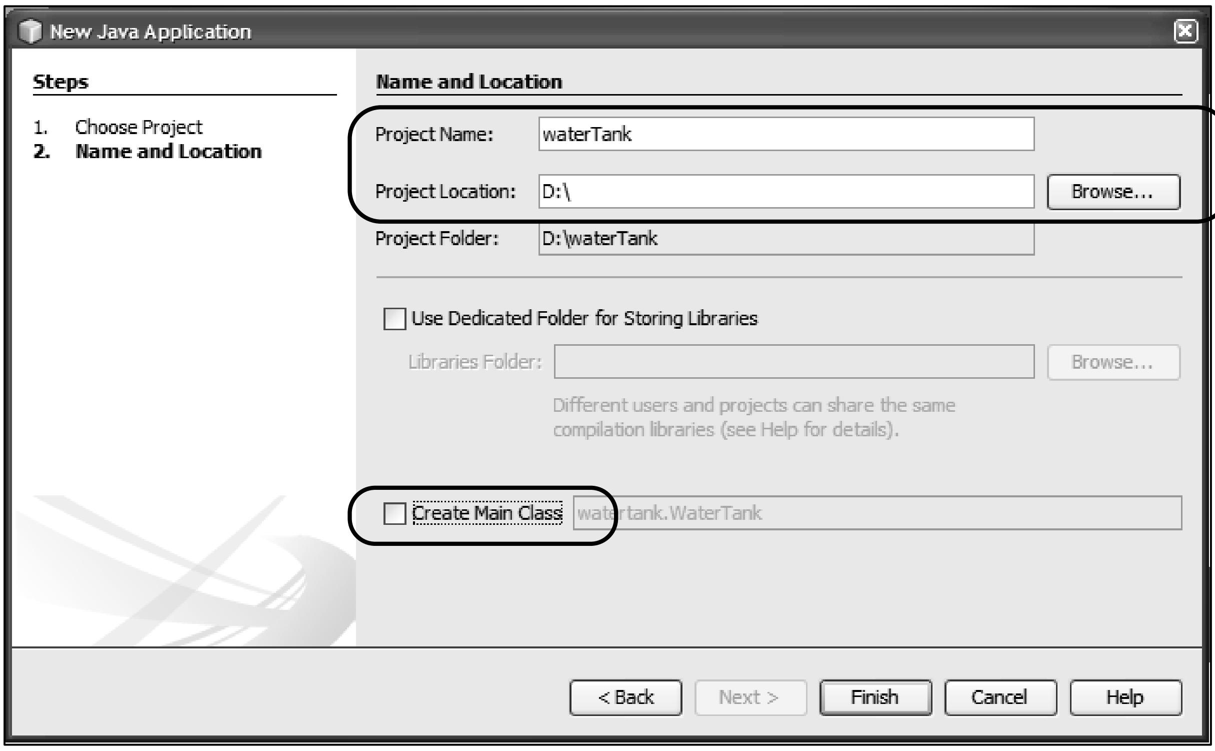
- length L
- width W
- height H

of the tank, then output:

- the maximum volume of water which the tank could hold
- the area of the sheet metal required to construct the tank.

Use the NetBeans ***File*** menu option to ***Close All Projects***, then click the ***New Project*** option.

Check that ***Java*** / ***Java Application*** is selected then click the ***Next*** button. Create a project with the name ***waterTank***. Use the ***Browse*** button to choose the location where you want the project to be stored. Make sure that the ***Create Main Class*** option is not ticked:



Project Name: waterTank

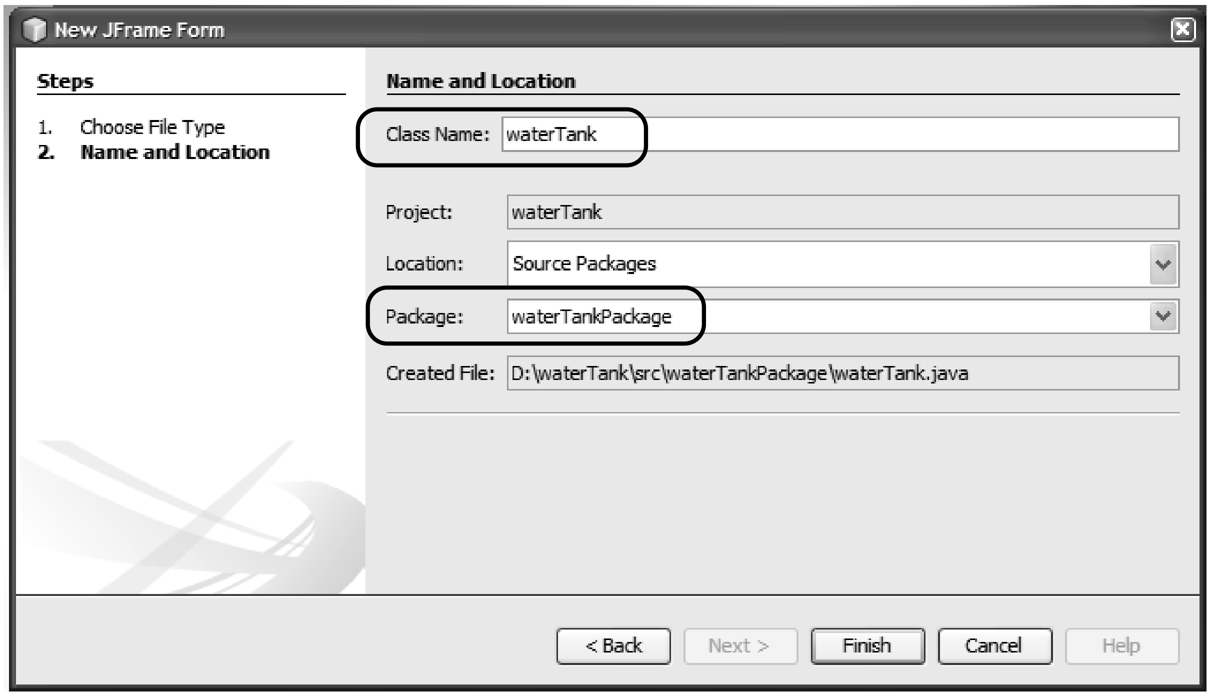
Project Location: D:\

Project Folder: D:\waterTank

☐ Use Dedicated Folder for Storing Libraries

☐ Create Main Class watertank.WaterTank

Return to the NetBeans editing page. Click the ***Projects*** tab at the left of the page, then right click on the ***waterTank*** project icon to open a drop-down menu. Select ***New***, then ***JFrame Form***. Set the ***Class Name*** as ***waterTank***, and the ***Package*** as ***waterTank Package***:



Class Name: waterTank

Project: waterTank

Location: Source Packages

Package: waterTankPackage

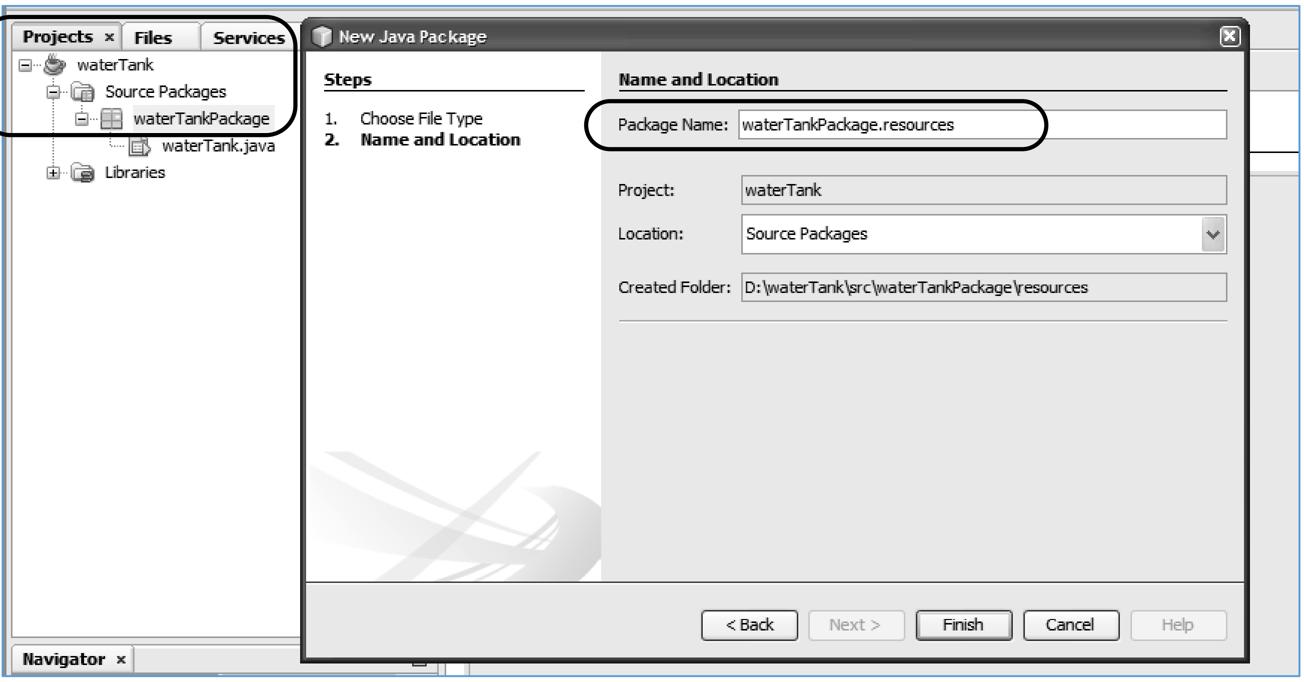
Created File: D:\waterTank\src\waterTankPackage\waterTank.java

Return to the NetBeans editing page. Before beginning work on the application, we will carry out the three steps necessary to ensure the correct behaviour of the program:

- 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.

This program again uses a picture image, so we will create a resources folder. Go to ***waterTank*** in the ***Projects*** window. Right-click on ***waterTankPackage*** and select ***New*** / ***Java Package***. Give the ***Package Name*** as ***waterTankPackage.resources***:



Package Name: waterTankPackage.resources
Project: waterTank
Location: Source Packages
Created Folder: D:\waterTank\src\waterTankPackage\resources

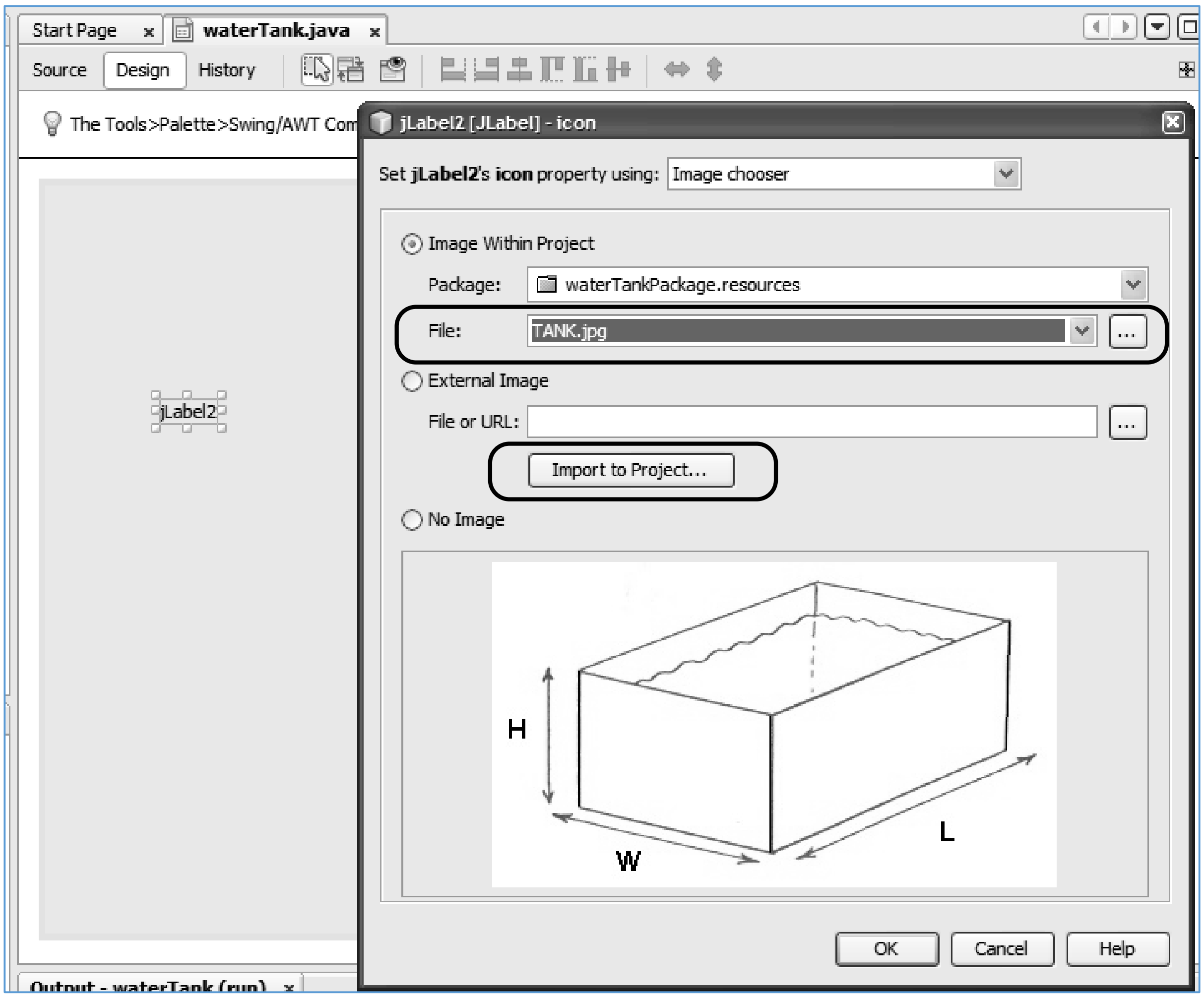
Return to the NetBeans editing page. Click the ***Design*** tab to move to the form layout view.

Before continuing, obtain or create a diagram of a water tank similar to the diagram at the start of this section. Save this in JPG format.

Select the ***Label*** component in the ***Palette***, then drag and drop a label onto the form.

Go to the ***Properties*** window and locate the ***icon*** property. Click the ellipsis (…) symbol at the right of the icon property line.

Use the ***Import to Project*** button to upload the diagram image. Check that this is selected on the ***File*** line:



Set **jLabel2**'s **icon** property using: Image chooser

Image Within Project

Package: waterTankPackage.resources

File: TANK.jpg

External Image

File or URL:

Import to Project...

No Image

Return to the NetBeans editing page. The image of the water tank should be displayed. If necessary, drag the image into the correct position on the form.

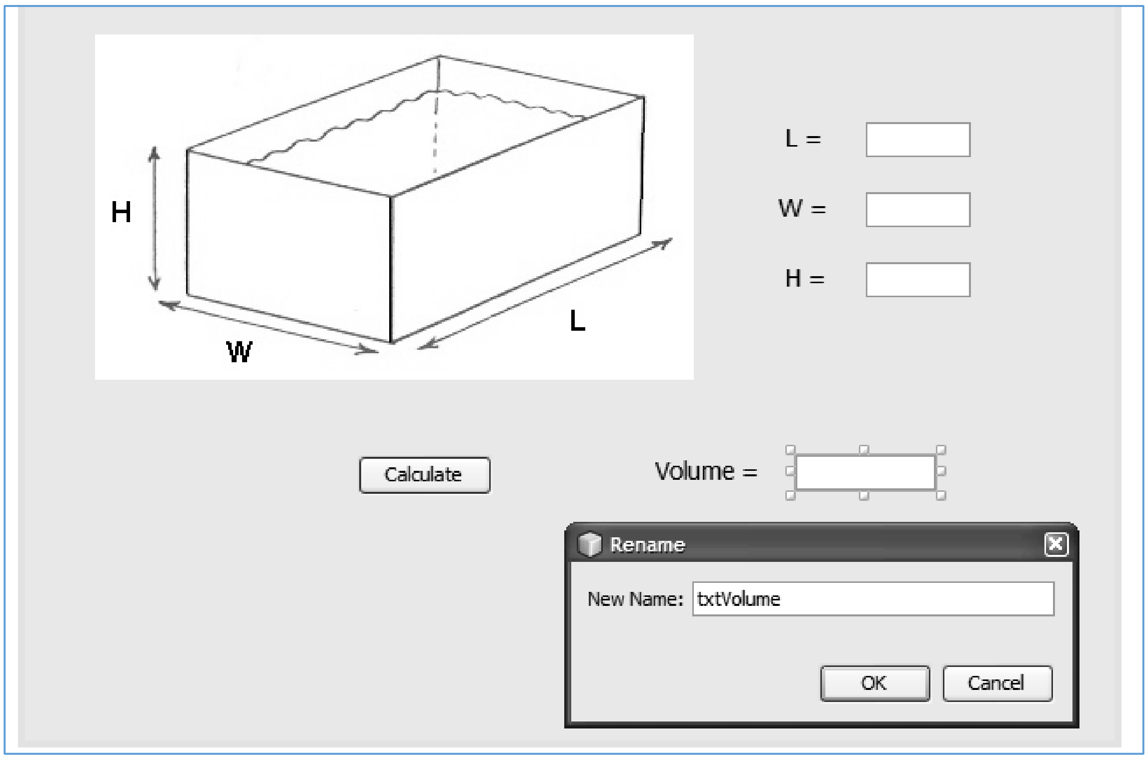
Right-click on the image and select ***Edit Text***. Delete the text caption from the label.

We can now work on the first stage of the calculation, to find the maximum volume of water which can be held in the tank. This will be given by the formula:

volume = length * width * height

Set up labels, text fields and a ***Calculate*** button as shown below. Rename the button as ***btnCalculate***. If necessary, drag the form wider so that there is space for all the components. Give names to the text fields:

txtLength
txtWidth
txtHeight
txtVolume



New Name: txtVolume

Click the ***Source*** tab to show the program code. Add a line near the top of the program to load extra code to create a message box:

```
package waterTankPackage;

import javax.swing.JOptionPane;

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

Just below, at the start of the ***waterTank class***, add the variables:

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

    double L, W, H, volume, area;

    public waterTank() {
        initComponents();
    }
```

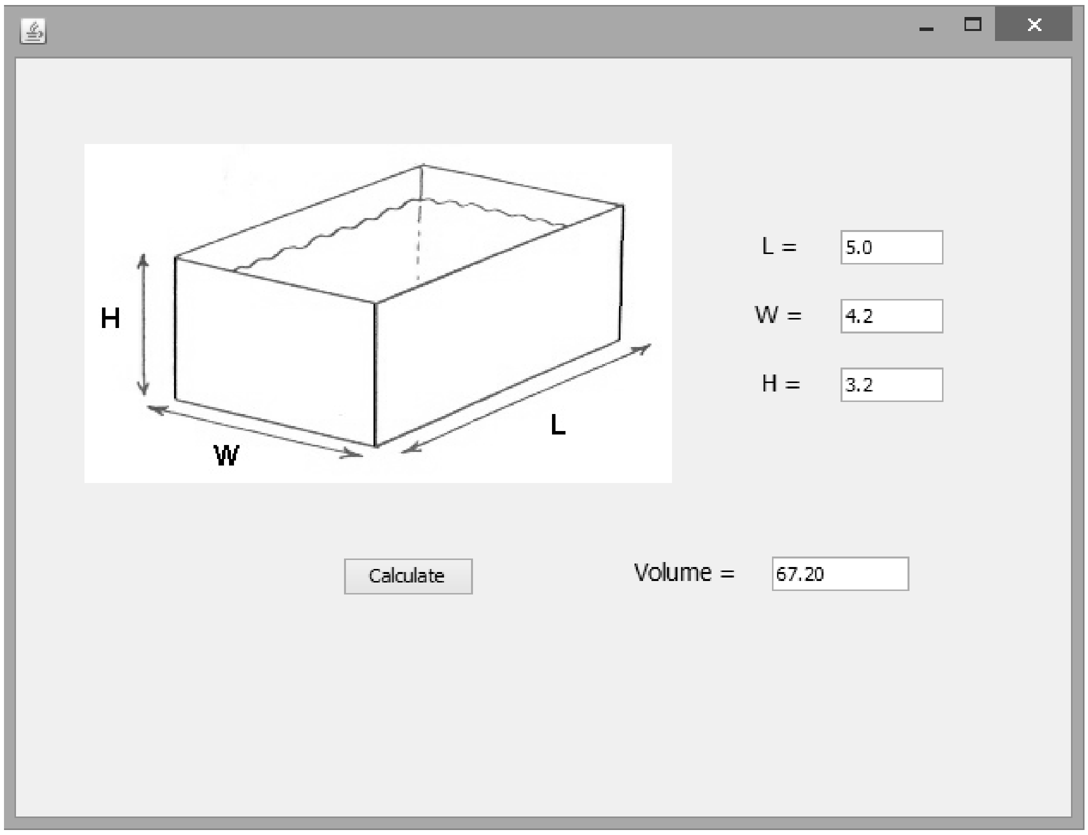
Click the ***Design*** tab to return to the design page.

Double click the ***Calculate*** button to create an empty ***method***. Add lines of program code:

```
private void btnCalculateActionPerformed(java.awt.event.ActionEvent evt) {
  try
  {
    L=Double.parseDouble(txtLength.getText());
    W=Double.parseDouble(txtWidth.getText());
    H=Double.parseDouble(txtHeight.getText());
    volume = L * W * H;
    txtVolume.setText(String.format("%.2f",volume));
  }
  catch(NumberFormatException e)
  {
     JOptionPane.showMessageDialog(waterTank.this, "Incorrect number format");
  }
 }
```

As in the previous program, we will output the answer to two decimal places, and use a ***try...catch*** block to display a message box if data is entered incorrectly.

Run the program. Use test data to check that the volumes of different sizes of tank can be calculated correctly. Verify the results independently using a calculator or spreadsheet.



L = 5.0

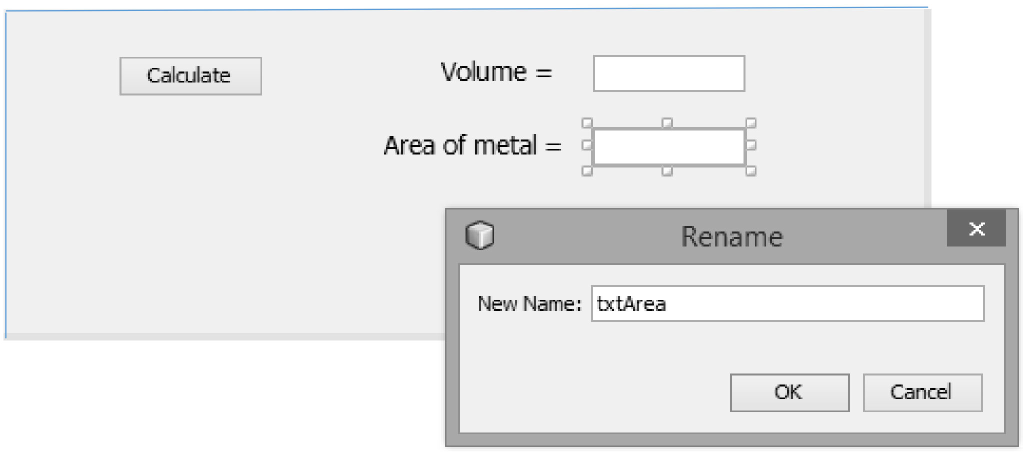
W = 4.2

H = 3.2

Volume = 67.20

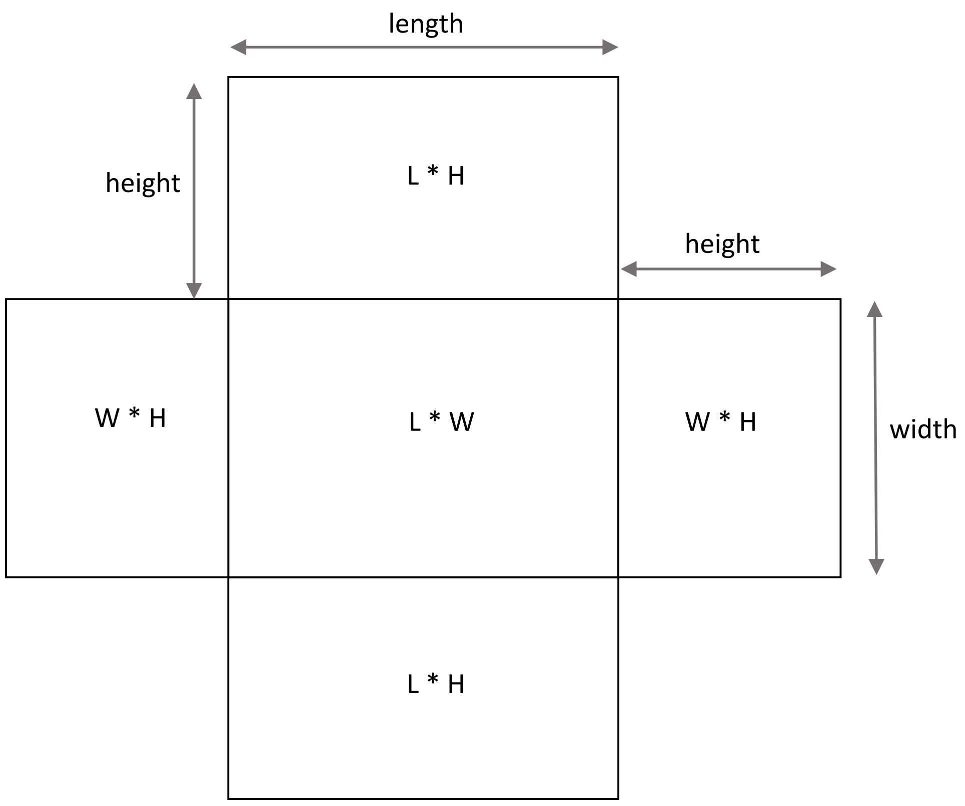
Close the program window and return to the NetBeans editing screen.

The final stage of the program is to calculate the area of metal required to construct the tank. Add a label and text field for displaying the result. Name the text field as ***txtArea***:



New Name: txtArea

We need to give some thought to how the area of metal should be calculated. Imagine the tank surface opened out into flat shape, known in mathematics as a ***net***:



From the diagram, we can see that the total area of metal required is the total of:

base	area = length * width
2 sides	area = length * height
2 sides	area = width * height

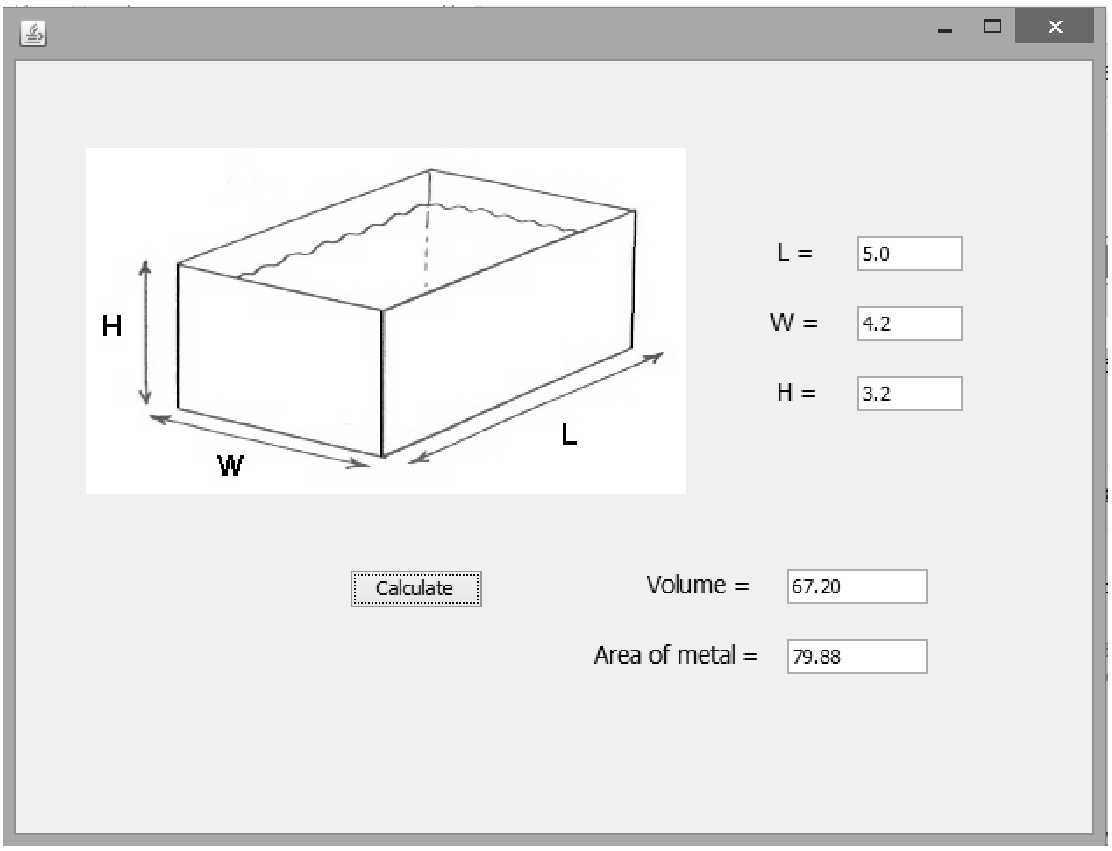
This could be written as a formula:

volume = (L * W) + 2 * (L * H) + 2 * (W * H)

Return to the ***btnCalculateActionPerformed*** method in the program listing. Add lines to carry out the area calculation and display the result in the ***txtArea*** text field:

```
private void btnCalculateActionPerformed(java.awt.event.ActionEvent evt) {
    try
    {
      L=Double.parseDouble(txtLength.getText());
      W=Double.parseDouble(txtWidth.getText());
      H=Double.parseDouble(txtHeight.getText());
      volume = L * W * H;
      txtVolume.setText(String.format("%.2f",volume));

      area = (L*W) + 2*(L*H) + 2*(W*H);
      txtArea.setText(String.format("%.2f",area));

    }
    catch(NumberFormatException e)
    {
       JOptionPane.showMessageDialog(waterTank.this, "Incorrect number format");
    }

}
```

Run the program and check that area of metal can be calculated correctly for different sizes of tank:

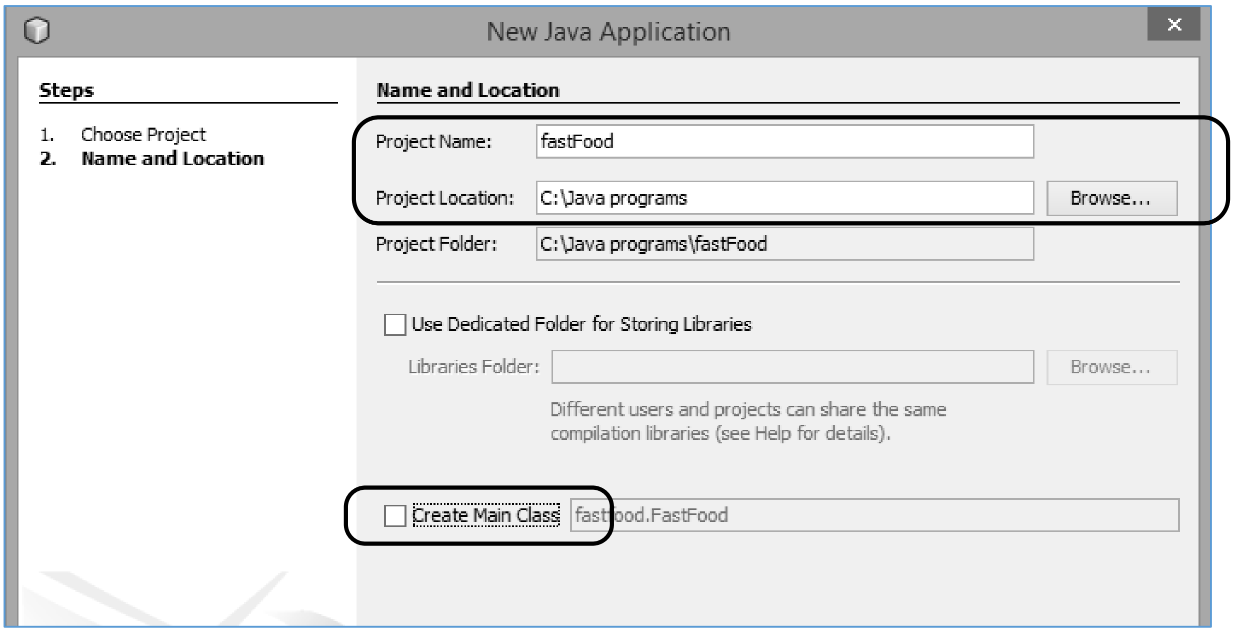


L = 5.0
W = 4.2
H = 3.2
Volume = 67.20
Area of metal = 79.88

For the final program in this section, we will create an application for use in a Fast Food Restaurant, to calculate the total cost of customers' orders.

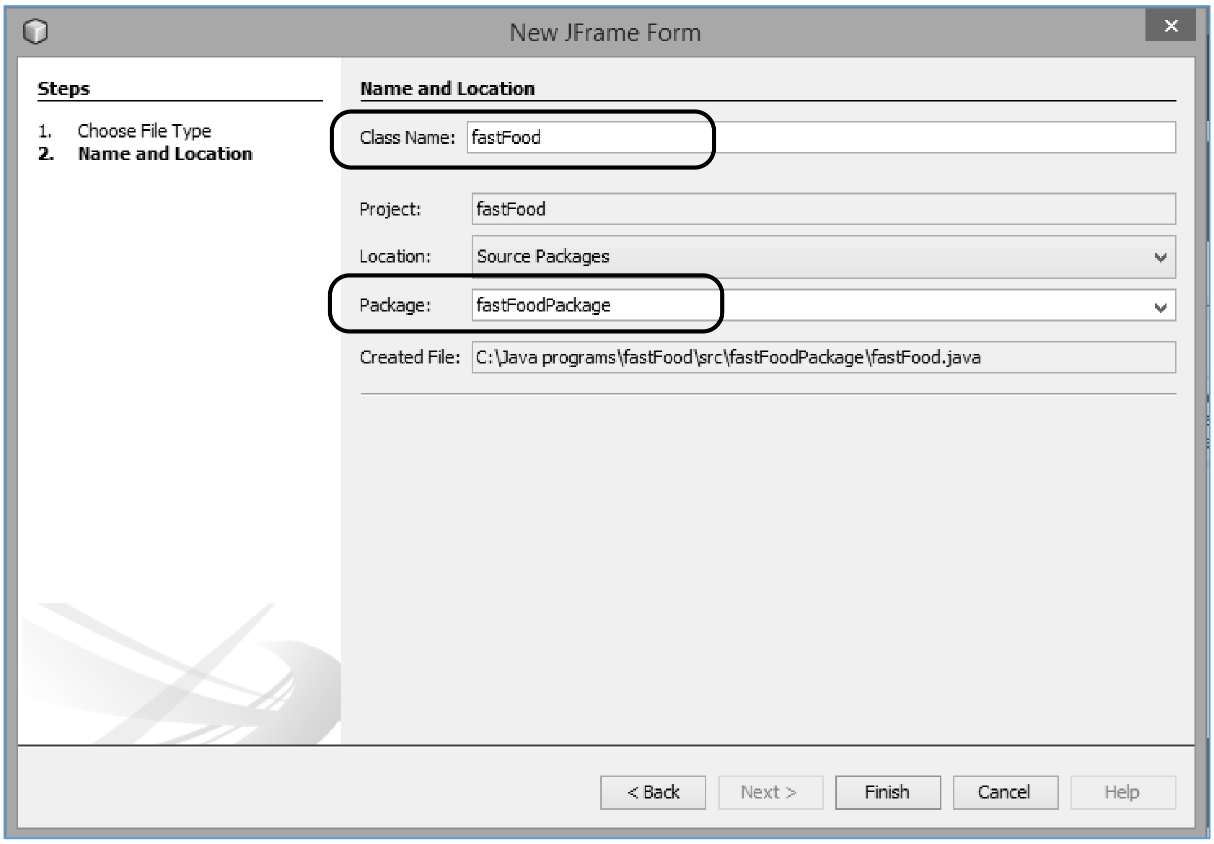
Close all current projects, then select ***New Project*** from the ***File*** menu of the NetBeans editor.

Set the ***Project Name*** to ***fastFood***, and ensure that ***Create Main Class*** is not ticked:



Project Name: fastFood
Project Location: C:\Java programs
Project Folder: C:\Java programs\fastFood
☐ Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).
☐ Create Main Class fastFood.FastFood

Return to the NetBeans editing page. Click the ***Projects*** tab at the top left of the page, then right-click on the ***fastFood*** project icon to open a drop-down menu. Select ***New***, then ***JFrame Form***. Set the ***Class Name*** as ***fastFood***, and the ***Package*** as ***fastFoodPackage***:



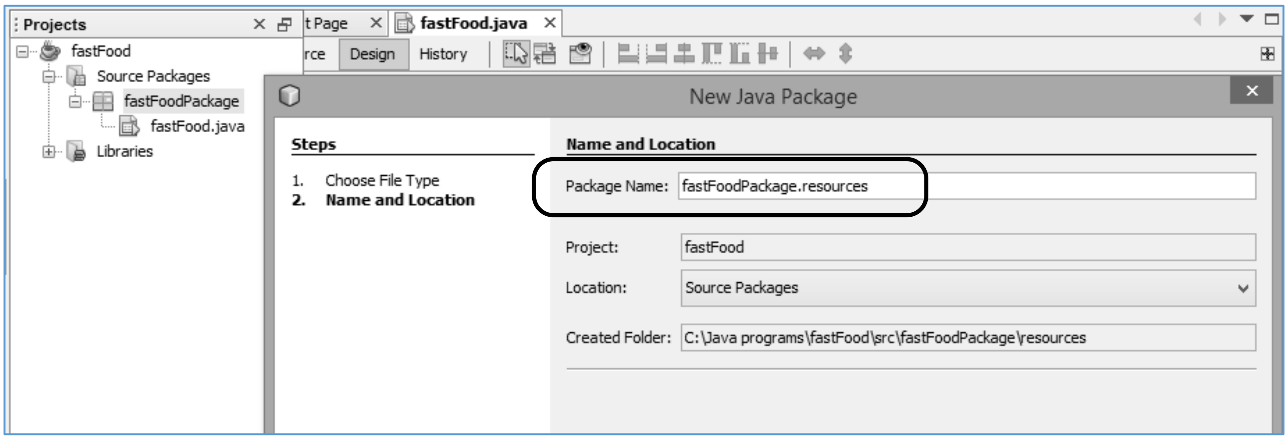
Class Name: fastFood
Project: fastFood
Location: Source Packages
Package: fastFoodPackage
Created File: C:\Java programs\fastFood\src\fastFoodPackage\fastFood.java

Click the ***Finish*** button to return to the NetBeans editor page.

- 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.

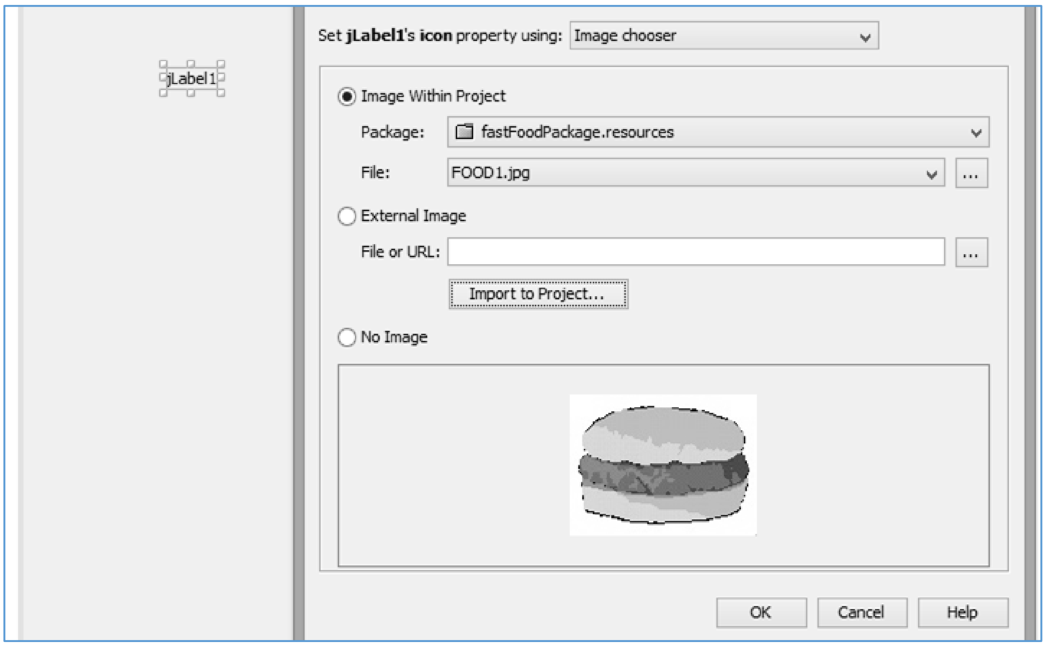
This program will again use picture images, so we will create a resources folder. Open the ***fastFood*** project structure until ***fastFoodPackage*** is reached. Right-click on fastFoodPackage and select ***New*** / ***Java Package***. Give the ***Package Name*** as ***fastFoodPackage.resources***:



Package Name: fastFoodPackage.resources
Project: fastFood
Location: Source Packages
Created Folder: C:\Java programs\fastFood\src\fastFoodPackage\resources

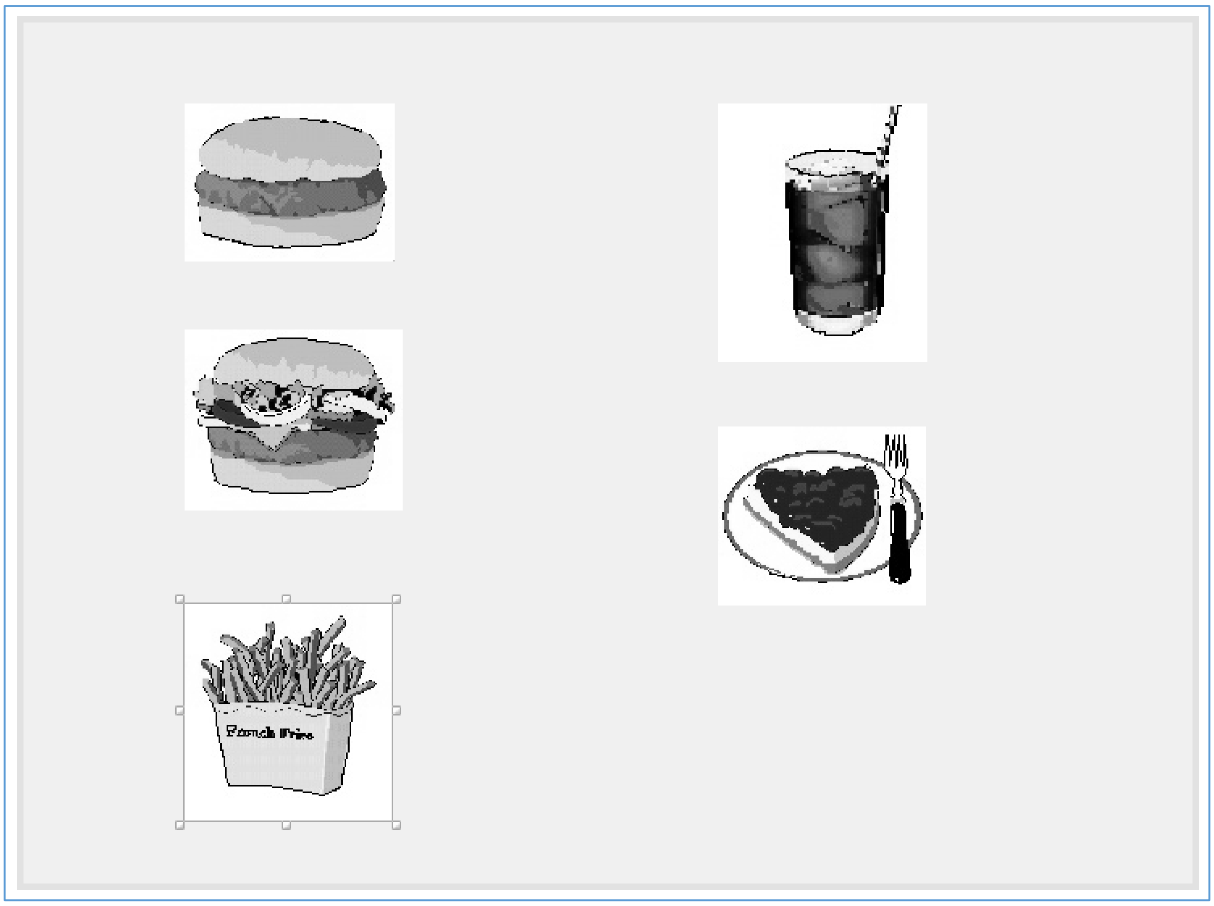
Before continuing, obtain picture images of five items which might be sold in a fast food restaurant, such as: burgers, fries, soft drink and cheesecake.

Add a ***label*** to the ***form***. Go to the ***Properties*** window and locate the ***icon*** property. Click the ellipsis (" ***…*** "), then use the ***Import to Project*** button to upload the five images. Select one of the image names on the ***File*** line:

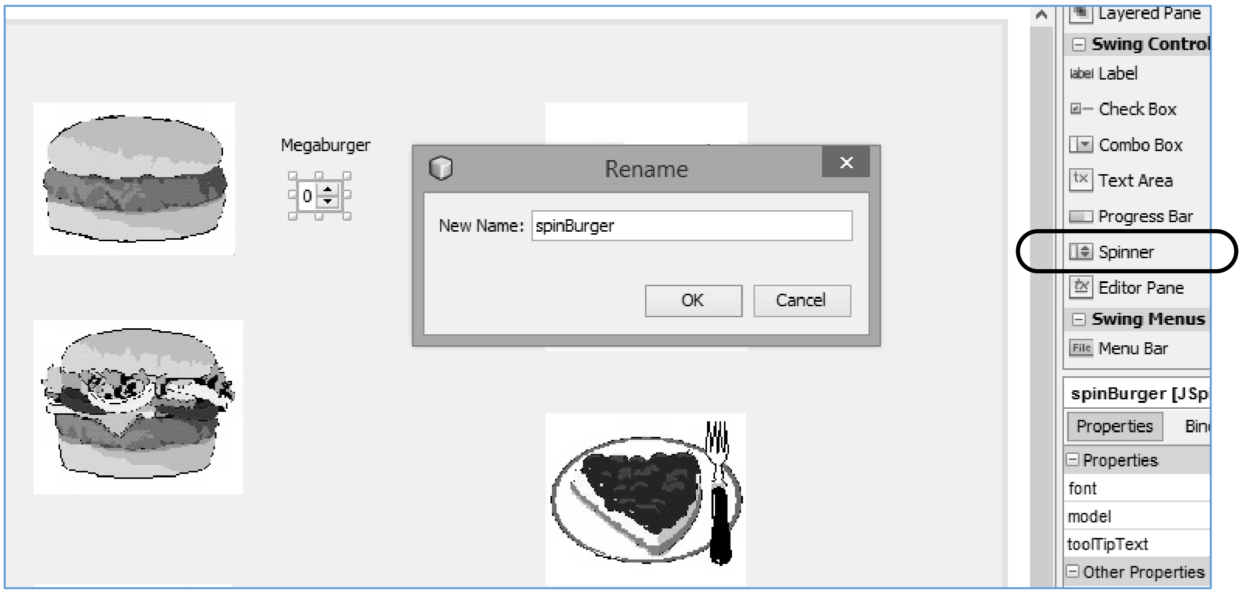


Set jLabel1's icon property using: Image chooser
(●) Image Within Project
Package: fastFoodPackage.resources
File: FOOD1.jpg
( ) External Image
File or URL:
( ) No Image

Return to the ***form*** and add four more ***labels***. Link a picture to each label, and delete the label text:

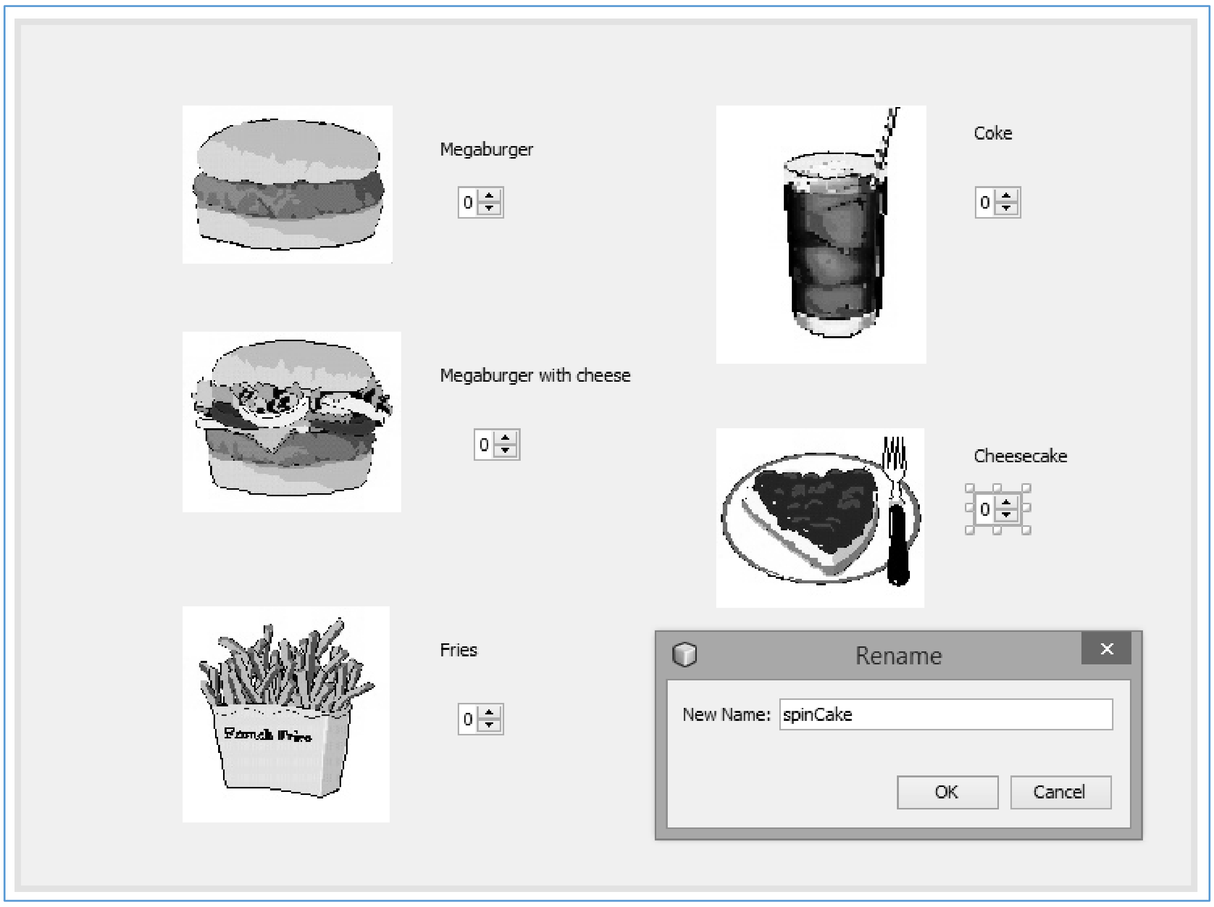


Locate the ***Spinner*** component in the ***Palette***. Drag and drop a spinner alongside the first image. Right-click on the spinner and rename the component as ***spinBurger*** or a similar suitable name.



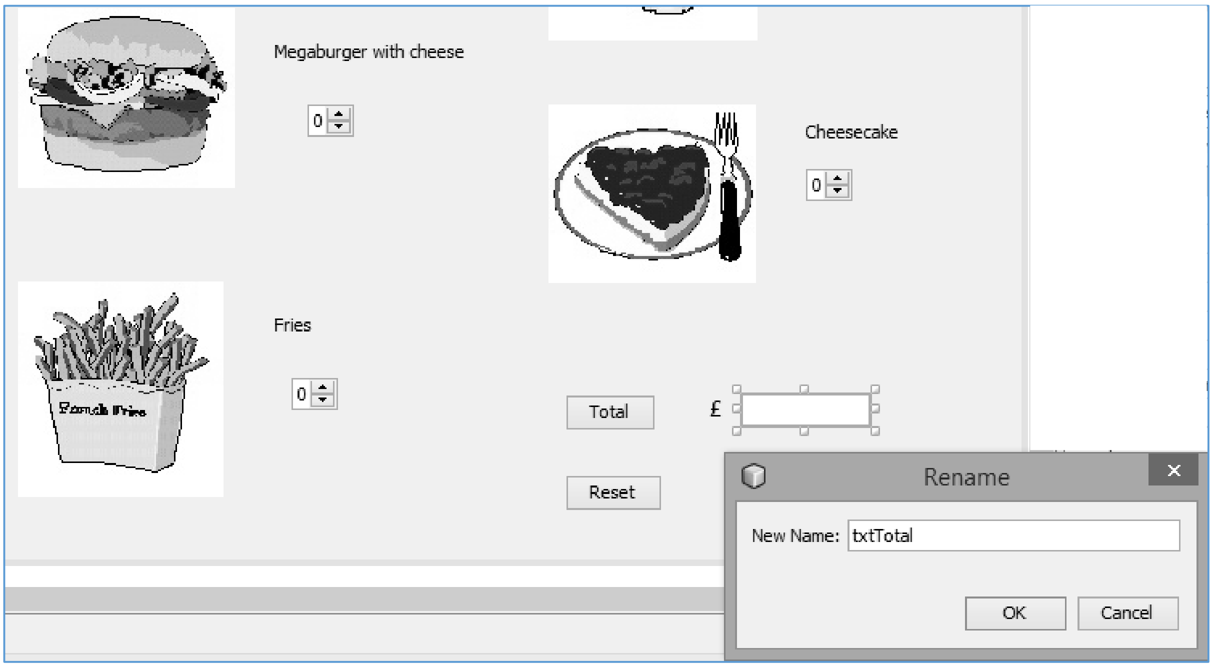
New Name: spinBurger

Add ***spinners*** in a similar way for the remaining pictures, renaming these as: ***spinBurgerCheese***, ***spinFries***, ***spinCoke*** and ***spinCake***. Also add a label for each menu item above the spinner:



New Name: spinCake

Complete the form by adding a ***button***, '**£**' ***label***, and ***text field*** with the name ***txtTotal*** to display the order total. Rename the button as ***btnTotal***. Also add a ***Reset*** button, which can be used to cancel values shown by the spinner components. Rename the button as ***btnReset***.



New Name: txtTotal

Double click the ***Reset*** button and add lines of program code to the method:

```
private void btnResetActionPerformed(java.awt.event.ActionEvent evt) {

    spinBurger.setValue(0);
    spinBurgerCheese.setValue(0);
    spinFries.setValue(0);
    spinCoke.setValue(0);
    spinCake.setValue(0);
    txtTotal.setText("");

 }
```

Run the program. Accept the ***main*** method offered.

Set numbers on each of the ***spinner*** components, then click the ***Reset*** button. check that all spinners are reset to zero.

Close the program window and return to the NetBeans editor.

The next stage is to calculate the cost of the customer's order. Begin by going to the ***Source*** page and setting up a series of ***constants*** at the start of the ***fastFood class***. Each line is a ***double*** number which records the price of a food item. These constants can then be used later in formulae to calculate the order cost.

```
package fastFoodPackage;

public class fastFood extends javax.swing.JFrame {

    double burgerPrice = 2.70;
    double burgerCheesePrice = 3.40;
    double friesPrice = 1.20;
    double cokePrice = 1.80;
    double cakePrice = 1.60;

    public fastFood() {
        initComponents();
    }
```

Using ***constants*** in this way is good programming practice. If the price of an item changes, the program only has to be changed in one easily accessible place, rather than the programmer having to search through the whole program for every separate occurrence of the original price.

Return to the ***Design*** page and double click the ***Total*** button to create an empty button click ***method***.

We will begin by adding code to calculate the cost of the first food item.

Set up a variable called ***total*** and initialise this to zero.

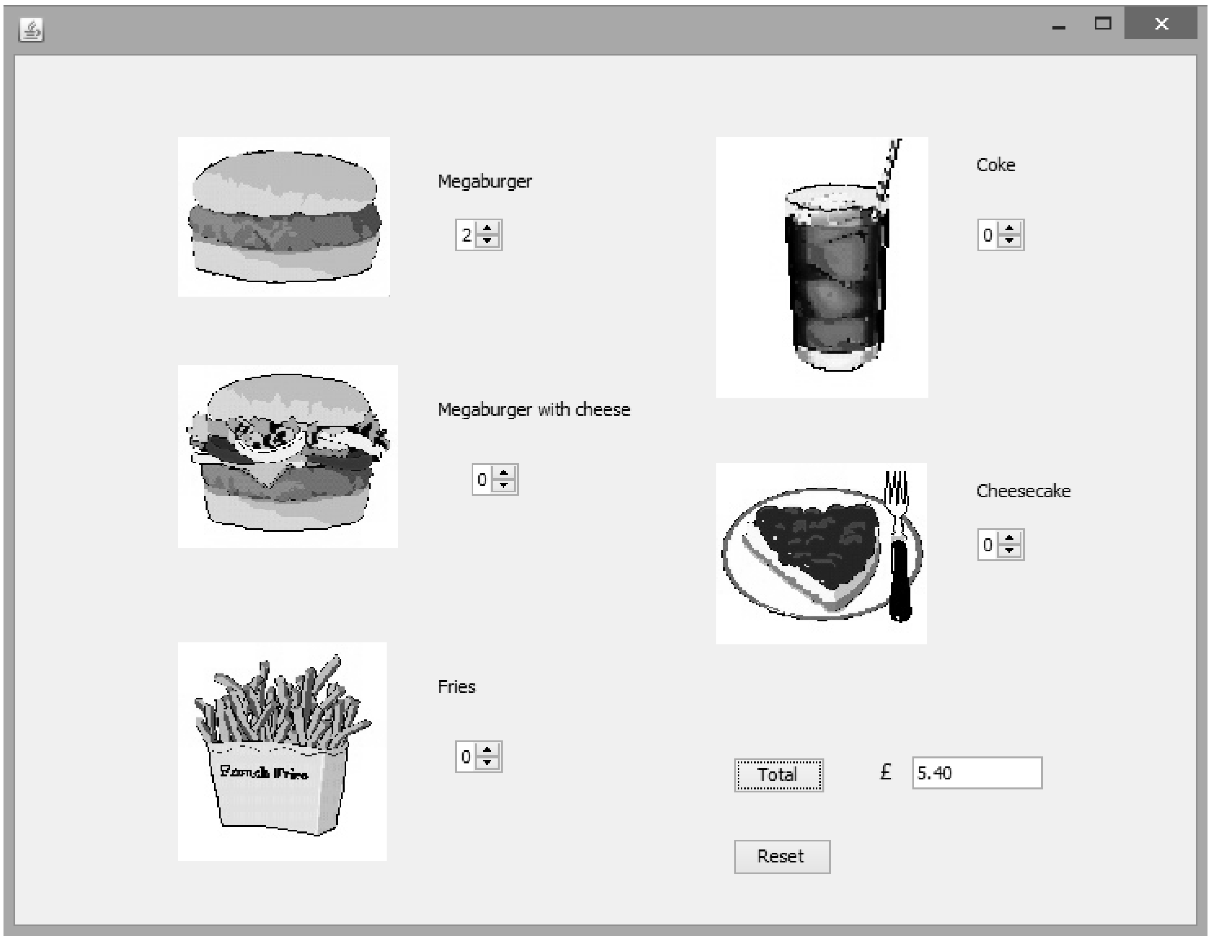
We can calculate the cost of the burgers purchased by:

(number purchased) * (price)

A simple way to add this amount to ***total*** is to use the '***+ =*** ' operator. The final result is then output to the ***txtTotal*** text field:

```
private void btnTotalActionPerformed(java.awt.event.ActionEvent evt) {
        double total=0;
        total += (Integer) spinBurger.getValue()*burgerPrice;

        txtTotal.setText(String.format("%.2f",total));
    }
```

Run the program. Select a quantity of the first food item, then click the Total button. Check that the correct order total is shown. You may find that the spinners are only wide enough to display single digits. If your restaurant is catering for larger groups, the spinner components can be dragged wider!

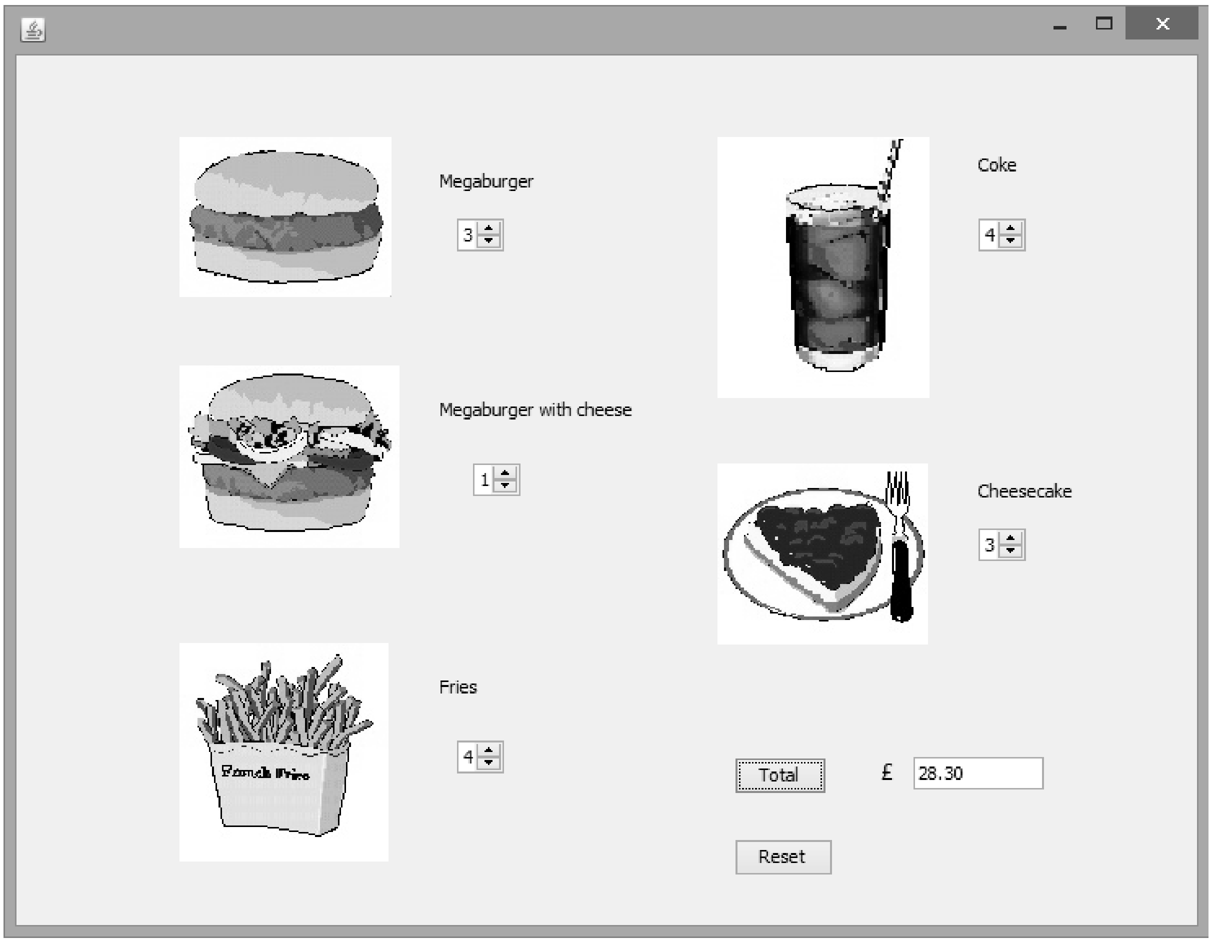


Close the program and return to the ***Source*** page. Locate the ***btnTotalActionPerformed*** method. Add further lines of code to calculate and add the cost of each of the remaining food items.

Be careful to specify the correct names for the ***price constants*** and ***spinners***:

```
private void btnTotalActionPerformed(java.awt.event.ActionEvent evt) {
     double total=0;
     total += (Integer) spinBurger.getValue()*burgerPrice;

     total += (Integer) spinBurgerCheese.getValue()*burgerCheesePrice;
     total += (Integer) spinFries.getValue()*friesPrice;
     total += (Integer) spinCoke.getValue()*cokePrice;
     total += (Integer) spinCake.getValue()*cakePrice;

     txtTotal.setText(String.format("%.2f",total));
  }
```

Run the program. Check that the totals for full orders are calculated correctly:


Megaburger
3
Megaburger with cheese
1
Fries
4
Coke
4
Cheesecake
3
Total
£
28.30
Reset