

5 Loops and arrays

In this chapter we will examine arrays, and the ways that they can be used in combination with loop structures to process data in programs.

The first project is a scientific application:

Monitoring of the level of pollutants in a river is carried out at regular intervals.

A program is required which will:

- input the number of readings collected
- input each of the readings and store these for processing.

The program should then output:

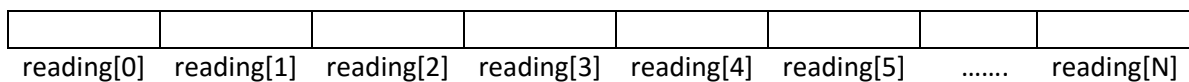
- a list of the readings
- the mean of the readings
- the maximum and minimum values recorded.

In this program we will need to store readings which are entered. To do this, variables will be required. We could set up individual variables for each reading, as we have done in previous programs:

```
int reading1;
int reading 2;
int readfing3;          etc...
```

However, this would be very tedious, and probably impossible if there was a very large number of readings. A better approach is to use an **array**. This acts like a series of numbered storage boxes. The whole array is given a name, and each individual box, or **element**, is given an **index number**. Arrays in Java always begin with **element 0**:

reading

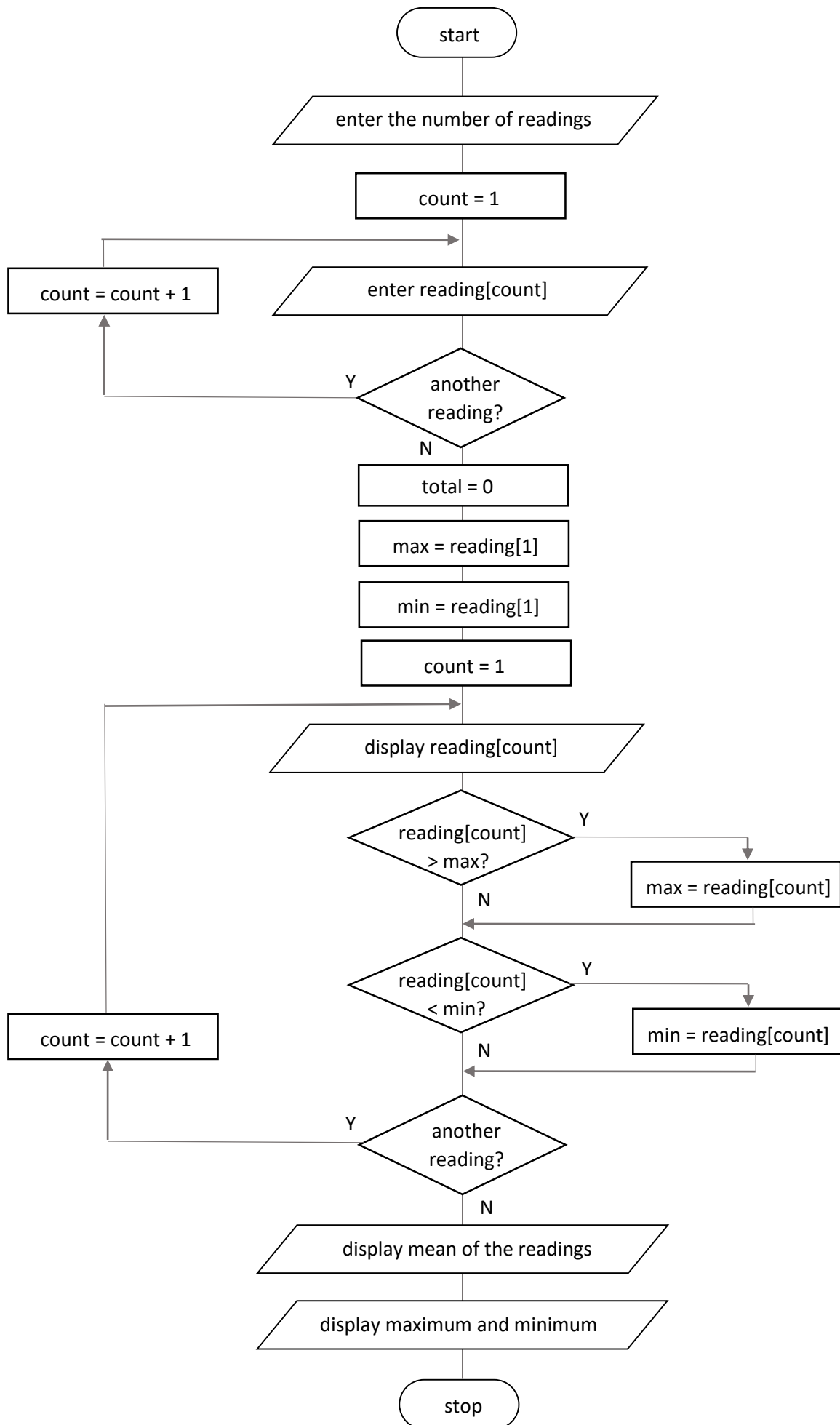


We set up the array with a line of program such as:

```
int[ ] reading = new int[100];
```

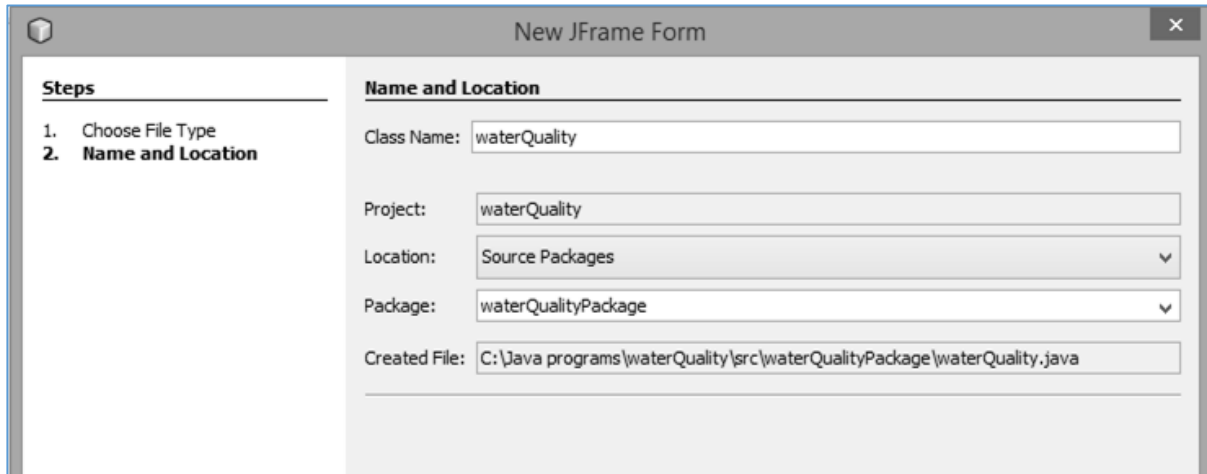
In this case, we have specified that the array should have elements for storing **integer** whole numbers, it is to be called "**reading**", and it is to have **100 elements** (which will be numbered 0 to 99).

A design for the program is given in the flowchart on the next page.



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

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

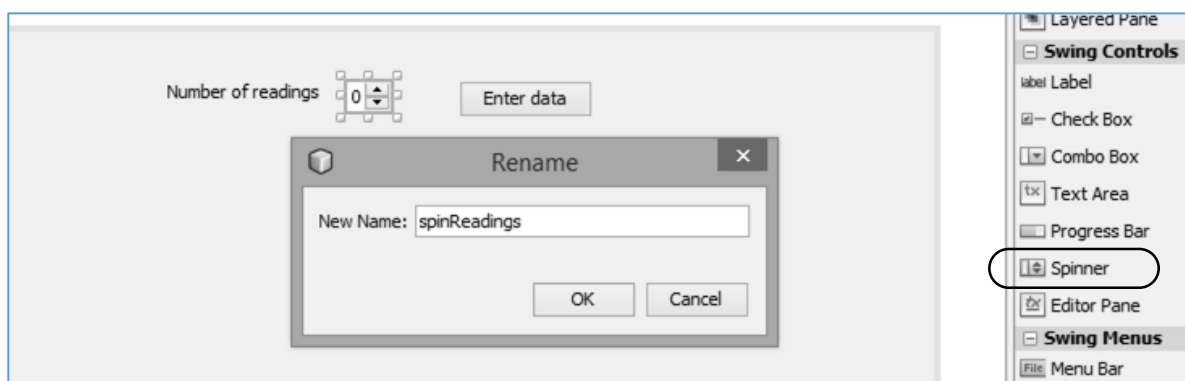


Click the **Finish** button to return to the NetBeans editing screen.

- 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**".

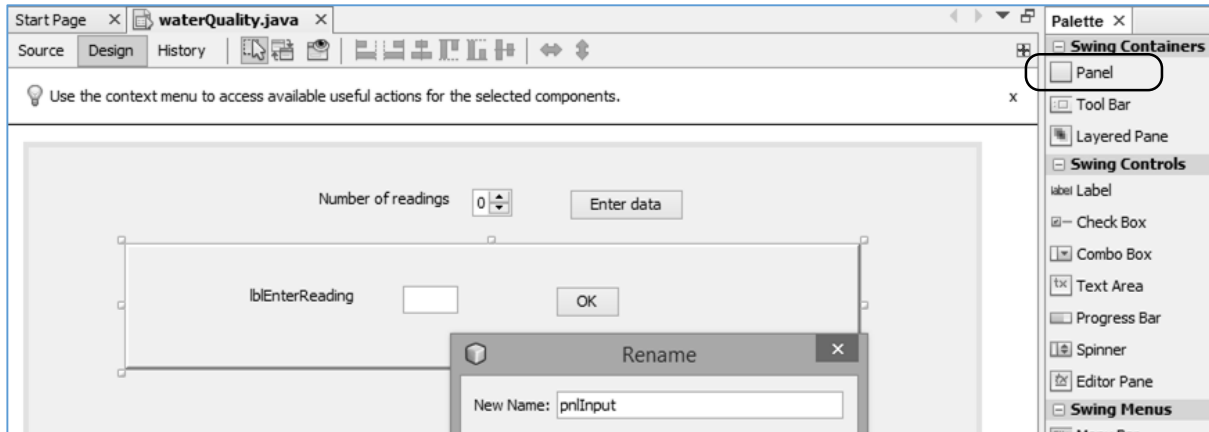
Run the program and accept the **main** class which is offered. Check that a blank window appears and has the correct size and colour scheme. Close the program and return to the editing screen. Click the **Design** tab to move to the form layout view.

Locate the **Spinner** component in the palette, and drag and drop this on the form. Rename the component as **spinReadings**. Add a **label** with the caption "**Number of readings**", and a **button** with the caption "**Enter data**". Rename the button as **btnStart**.

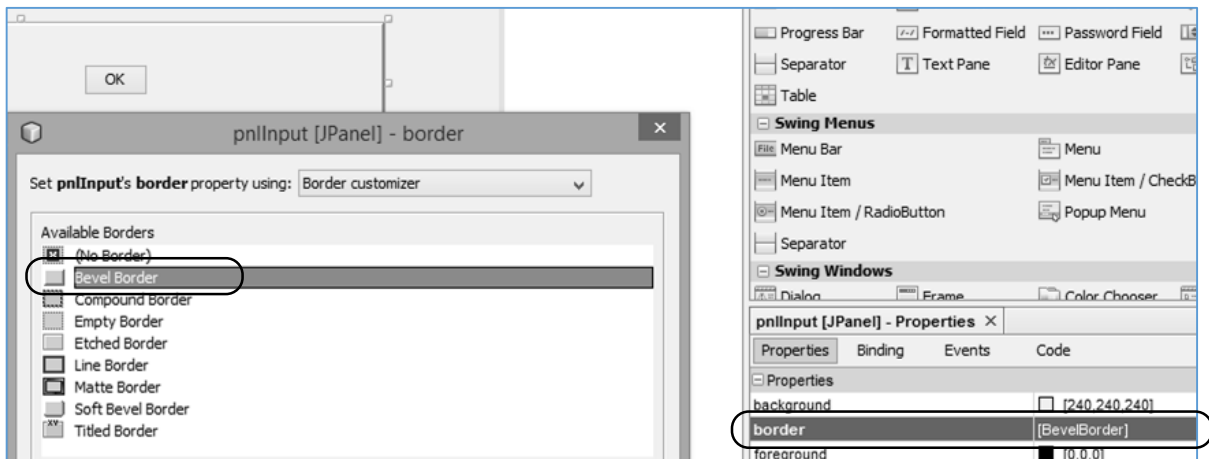


Locate the **Panel** component in the palette, and drag and drop this on the form. Right-click on the panel and select **Layout / Absolute Layout**.

- Rename the panel as **pnIInput**.
- Add a **label** to the panel and rename this as **lblEnterReading**. (The caption for this label will be set by the program when it runs, so the default text for the label need not be changed.)
- Add a **text field** with the name **txtReading**, and a **button** with the name **btnEnter**.



Select the panel and locate the **border** row in the property window. Click in the right hand column to open the **border style window**, and select **Bevel Border**:



Click the **Source** tab to open the code window. We will first set up an **array** to store the readings, allowing for a maximum of 100 integer values to be entered. Add variables to record the total number of readings to be entered, and a count of the number of readings entered so far. We will set the **input panel** to **not be visible** when the program first runs:

```
public class waterQuality extends javax.swing.JFrame {
    int[] pollutant = new int[100];
    int totalReadings;
    int count;

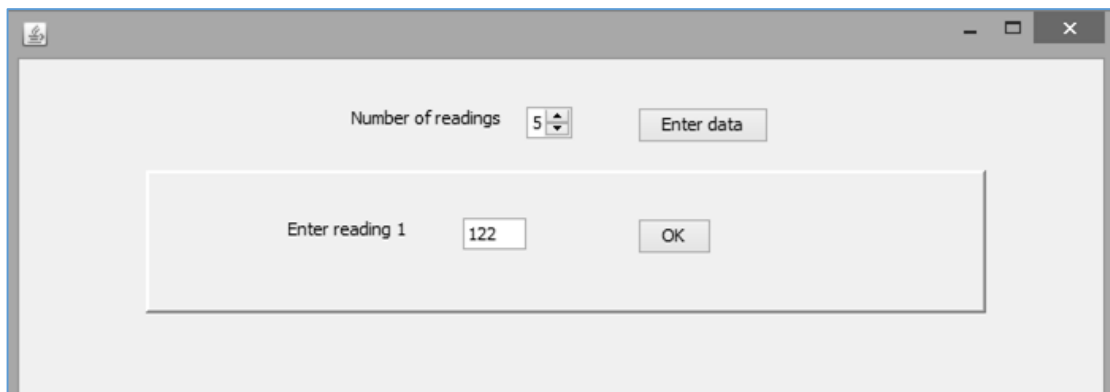
    public waterQuality() {
        initComponents();
        pnIInput.setVisible(false);
    }
}
```

Click the **Design** tab to return to the form, then double click the "**Enter data**" button alongside the spin component to create a **method**. Add lines of code to the method which will:

- collect the number of readings from the spin component
- make the input panel visible
- initialise the count of readings entered to 1
- set the label text on the panel to display the number of the reading which is currently being entered.

```
private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {
    totalReadings=(Integer) spinReadings.getValue();
    pnlInput.setVisible(true);
    count=1;
    lblEnterReading.setText("Enter reading "+ Integer.toString(count));
}
```

Run the program. Set the number of readings using the spin component, then click the '**Enter data**' button. Check that the panel appears, the caption '**Enter reading 1**' is displayed, and a value can be entered in the text field. You may need to adjust the size of the label and text field.



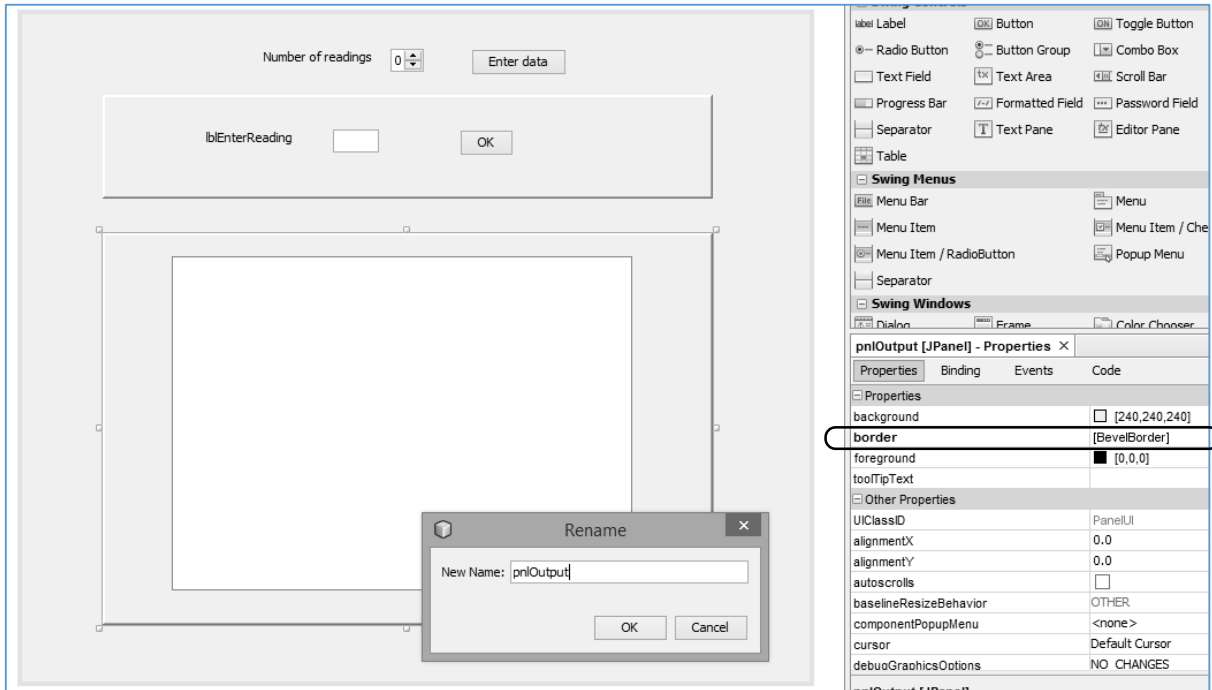
Close the program and return to the editing screen. Click the **Design** tab to go to the form, then double click the **OK button** to create a **method**. Add code which will:

- collect the value which has been entered in the text field, and store it in the array,
- increase the count of readings by one,
- if all the data has not yet been entered, then change the label caption to display the number of the current reading:

```
private void btnEnterActionPerformed(java.awt.event.ActionEvent evt) {
    pollutant[count]=Integer.parseInt(txtReading.getText());
    txtReading.setText("");
    count++;
    if (count<= totalReadings)
    {
        lblEnterReading.setText("Enter reading "+ Integer.toString(count));
    }
}
```

We must now consider what will happen when data entry is completed. The specification asks for the readings to be displayed in a list, along with the mean, maximum and minimum values. To do this we will add another panel to the form. Click the **Design** tab to move to the form layout view, then drag and drop a **Panel** component below the input panel. Rename the panel as **pnlOutput**. Go to the **border** property line and select **Bevel Border**.

Add a **text area** to the panel and name this as **txtOutput**:



It would be best if this panel is not visible when the program begins. Click the **Source** tab to open the program code window. Locate the **waterQuality()** method, and add a line to set the **visible** property for the panel to **false**:

```
public waterQuality() {
    initComponents();
    pnlInput.setVisible(false);
    pnlOutput.setVisible(false);
}
```

Locate the **btnEnterActionPerformed** method which you began to develop earlier. We can now add lines of program code which will operate when the user enters the final reading. These lines of code will:

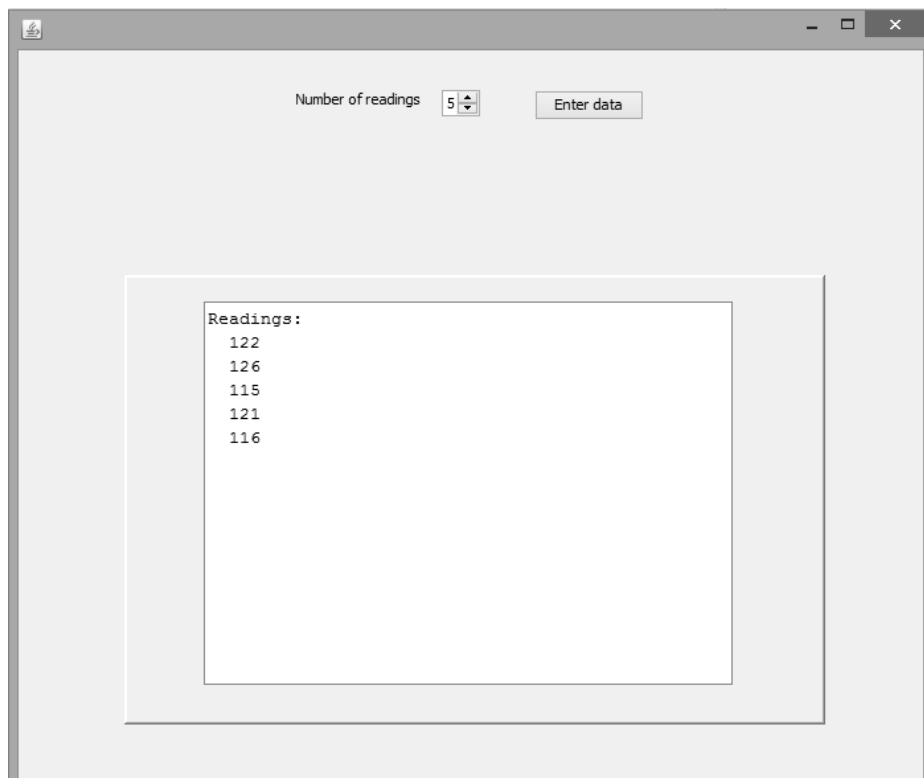
- make the **input panel** close and the **output panel** appear,
- define a **string variable 's'** which will be used to display the output text,
- use a **loop** to collect each of the readings from the **pollutant[]** array and add it to **s**, so that they will be displayed in the output list.

Add the **ELSE...** block to the method:

```
private void btnEnterActionPerformed(java.awt.event.ActionEvent evt) {
    pollutant[count]=Integer.parseInt(txtReading.getText());
    txtReading.setText("");
    count++;
    if (count<= totalReadings)
    {
        lblEnterReading.setText("Enter reading "+ Integer.toString(count));
    }
    else
    {
        pnlInput.setVisible(false);
        pnlOutput.setVisible(true);
        String s="";
        s+="Readings: \n";
        for (int i=1; i<=totalReadings; i++)
        {
            s+=" "+ Integer.toString(pollutant[i])+"\n";
        }
        txtOutput.setText(s);
    }
}
```

Run the program. Set the number of readings which will be entered, then click the "**Enter data**" button. The input panel should appear, prompting you to enter each of the readings in turn.

When the correct number of readings have been entered, the output panel should open and display a list of the readings:



Close the program and return to the Source code editing window.

Notice the structure of the loop control line:

```
for (int i = 1; i <= totalReadings; i++)
```

This is made up of three parts, separated by semi-colons. The first part:

```
int i = 1;
```

defines a **local variable** *i* which will be used as a **counter** while the loop is repeating. We are giving this variable an initial value of 1.

The next part:

```
i <= totalReadings;
```

tells the computer to keep repeating the loop as long as the value of *i* is **less than or equal to** the variable **totalReadings**. This will make the loop repeat for the correct number of data entries.

The final section:

```
i++
```

tells the computer to **add 1 to i** each time around the loop.

The next requirement of the program is to calculate the **mean** of the readings. This requires us to make a total of the values entered, then divide by the number of readings. The total can be calculated within the loop.

Add the lines of code to calculate and display the mean, using an accuracy of one decimal place:

```
else
{
    pnlInput.setVisible(false);
    pnlOutput.setVisible(true);
    String s="";
    s+="Readings: \n";

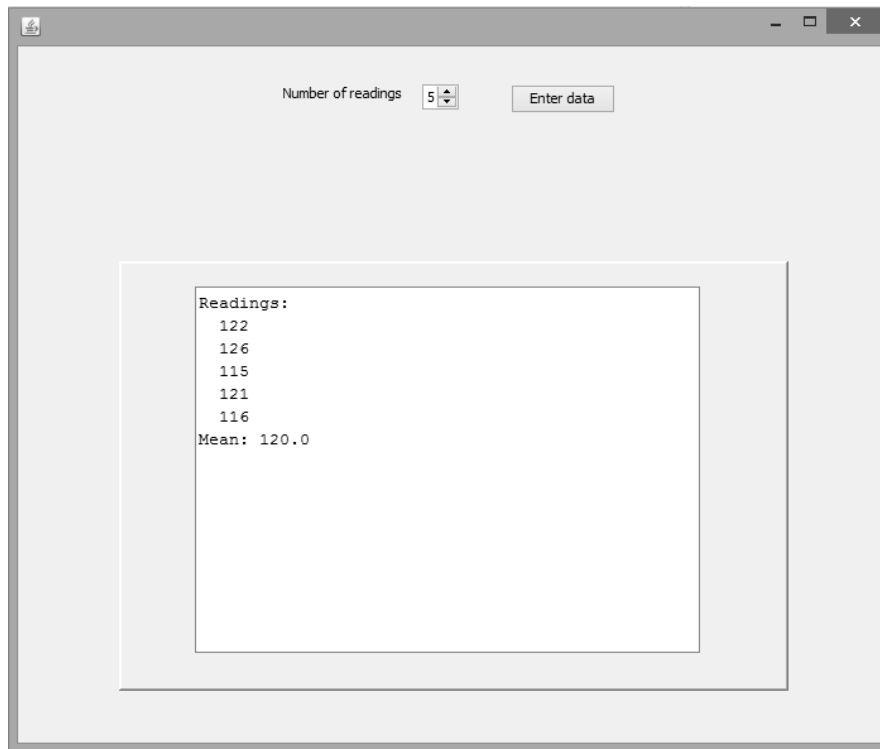
    double sum=0;
    double mean;

    for (int i=1; i<=totalReadings; i++)
    {
        s+=" "+Integer.toString(pollutant[i])+"\n";
        sum += pollutant[i];
    }

    mean = sum/totalReadings;
    s+="Mean: "+String.format("%.1f",mean)+"\n";

    txtOutput.setText(s);
}
}
```


Run the program and check that the mean of a series of readings is calculated correctly:



The final requirement of the program was to find and display the maximum and minimum of the values. To do this, we can again make use of the loop which checks each of the values in the `pollutant[]` array.

122	126	115	121	116
reading[1]	reading[2]	reading[3]	reading[4]	reading[5]

Maximum 122
Minimum 122

A strategy we can use is to begin by assuming the **first array element** contains the **maximum** and **minimum** value for the data set. The loop then checks each of the array elements in turn:

- If a higher value is found, this becomes the new maximum.
- If a lower value is found, this becomes the new minimum.

122	126	115	121	116
reading[1]	reading[2]	reading[3]	reading[4]	reading[5]

Maximum 122 → 126 → 126
Minimum 122 → 115 → 115

Add lines of program code to initialise the *maximum* and *minimum* to the *first array value*, then carry out updates when necessary as the loop checks each array element:

```
s+="Readings: \n";
double sum=0;
double mean;

int max=pollutant[1];
int min=pollutant[1];

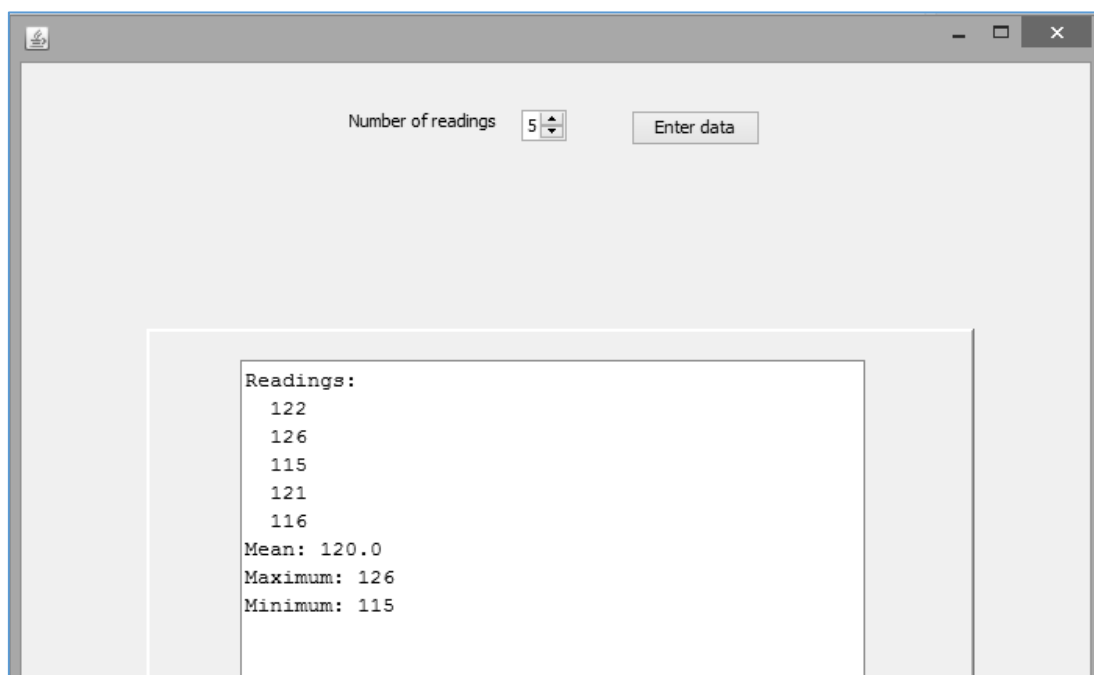
for (int i=1; i<=totalReadings; i++)
{
    s+=" "+Integer.toString(pollutant[i])+"\n";
    sum += pollutant[i];

    if (pollutant[i]>max)
    {
        max=pollutant[i];
    }
    if (pollutant[i]<min)
    {
        min=pollutant[i];
    }
}
mean = sum/totalReadings;
s+="Mean: "+String.format("%.1f",mean)+"\n";

s+="Maximum: "+Integer.toString(max)+"\n";
s+="Minimum: "+Integer.toString(min)+"\n";

txtOutput.setText(s);
}
```

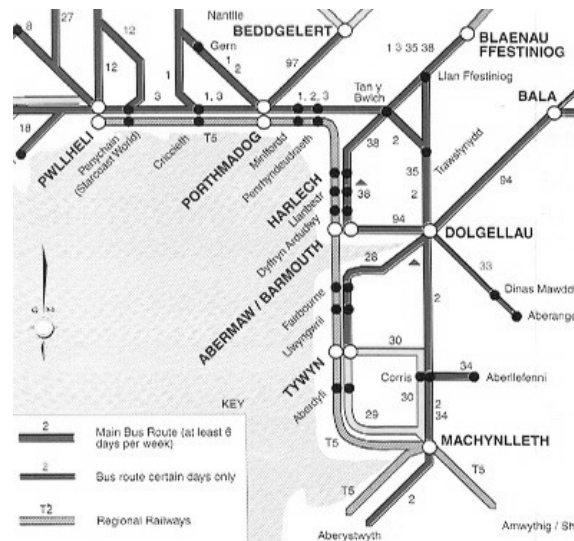
Run the program. Enter a set of test data and check that the maximum and minimum values are displayed correctly.



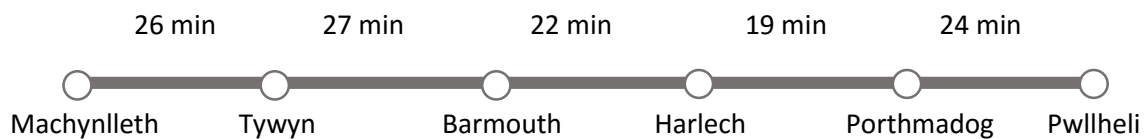
In the next program we will again make use of a loop and arrays, this time to produce a railway timetable:

A railway service runs along the coast of Cardigan Bay from Machynlleth to Pwllheli. A program is required which will produce a timetable for the journey.

The program should input the departure time from Machynlleth, then display the arrival times for other principal stations along the route.



Journey times between stations, including stops, are given below:



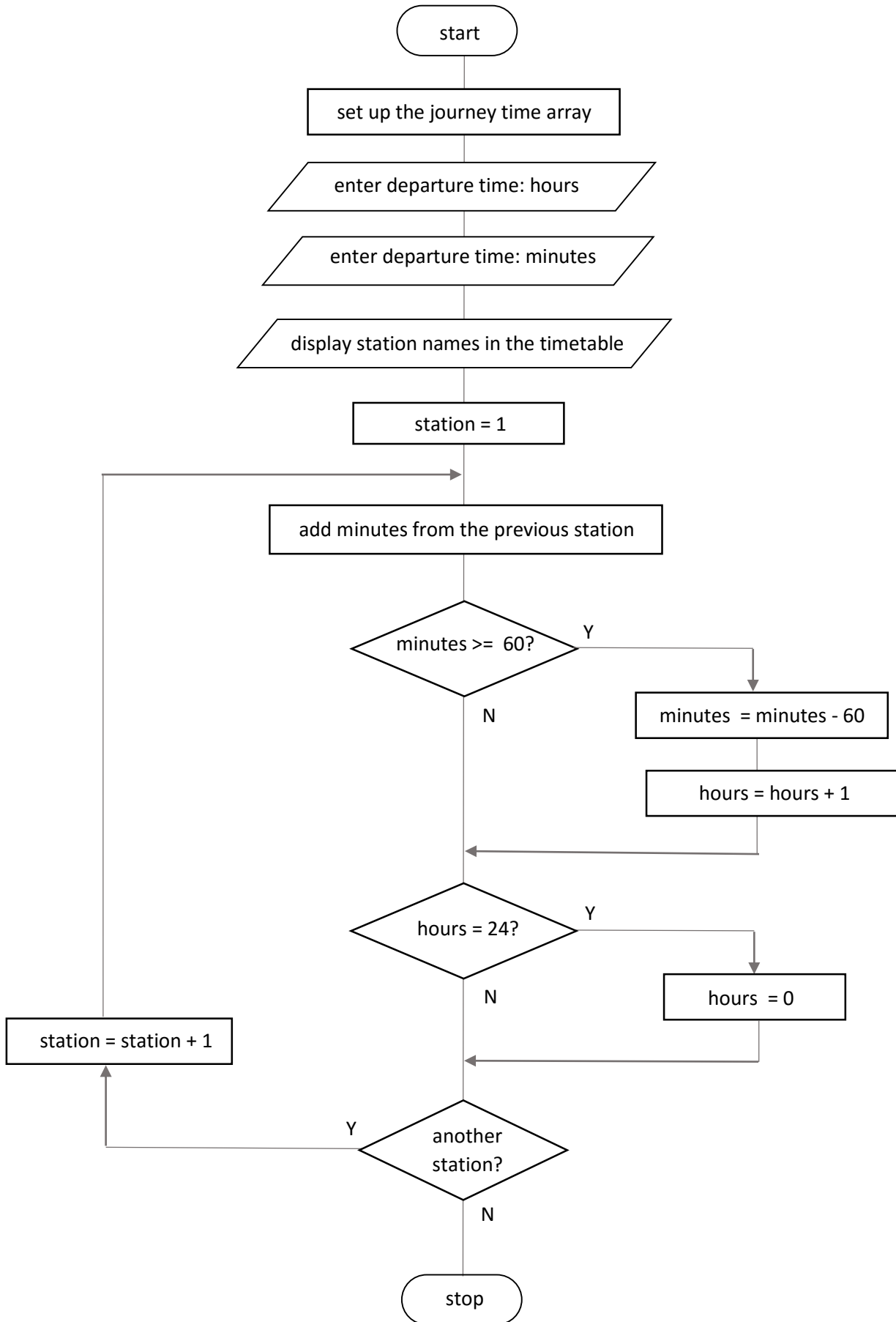
It will be convenient to store the journey times between stations in an array, so that a loop can be used to generate the timetable. A design for the program is given on the next page.

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

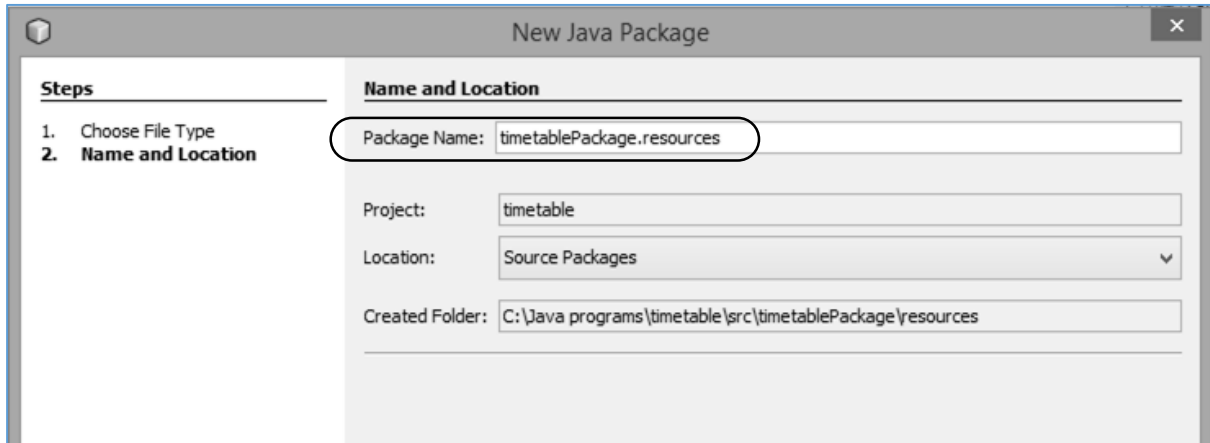
Click the **Finish** button to return to the NetBeans editing page. Right-click on the **timetable** project, and select **New / JFrame Form**. Give the **Class Name** as **timetable**, and the **Package** as **timetablePackage**. Click the **Finish** button to return to the NetBeans editing screen.

- 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 editing screen.

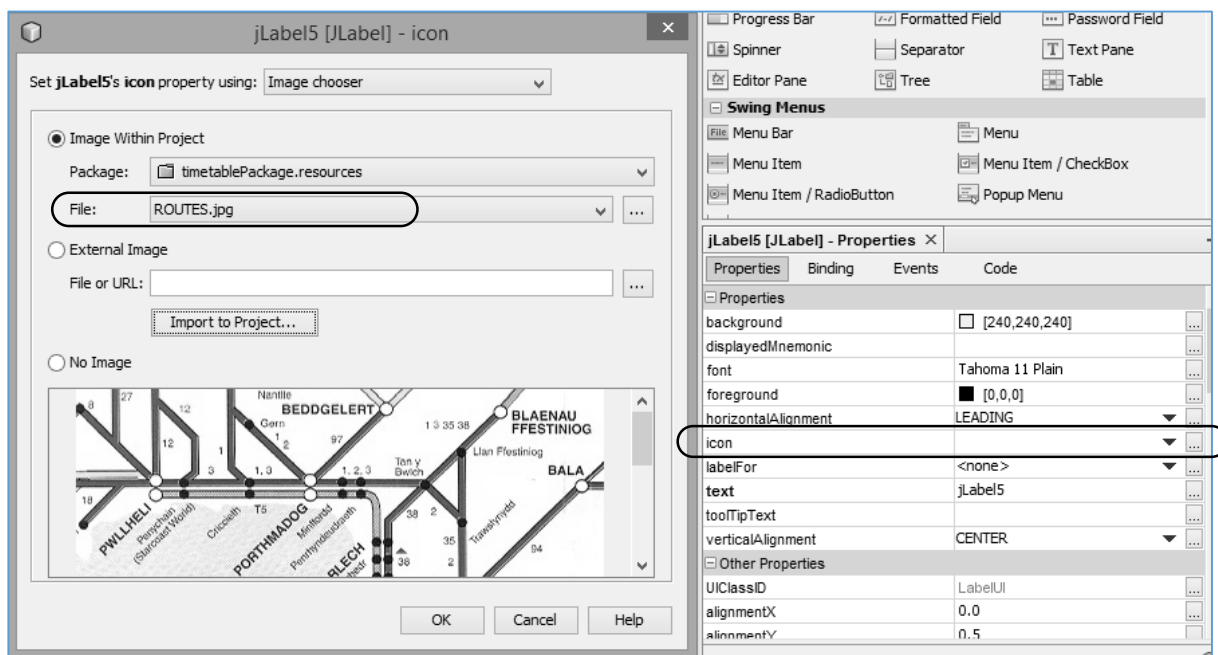


It will be helpful to users to see the route map for the journey, so we will display this as a picture image. Go to the **Projects** window and use the **+ icons** to open the **timetable** project folders until you reach **timetablePackage**. Right-click on **timetablePackage** and select **New / Java Package**. Set the **Package Name** to **timetablePackage.resources**:



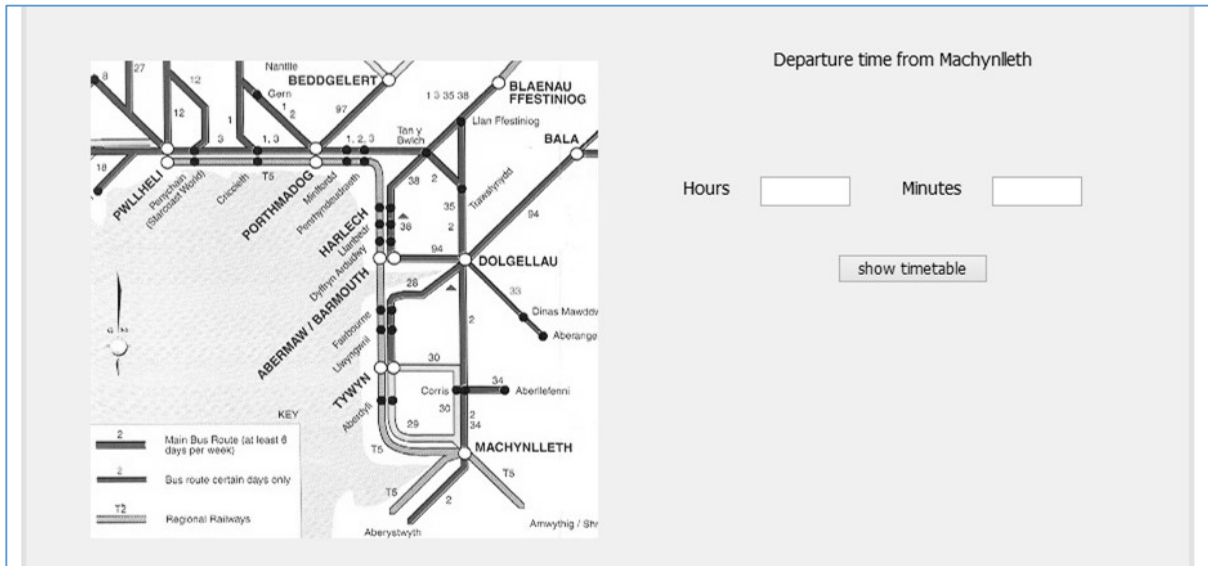
Before continuing, obtain or create a route map similar to the one shown above in the program specification. Use a graphics application such as Photoshop or Paint to set the width of the image to approximately 400 pixels.

Go to the NetBeans Design screen, and drag and drop a **label** component on the form. Locate the **icon** property, and click the ellipsis (" ... ") symbol to open the image selection window. Use the **Import to Project** button to upload the route map. Ensure that the name of this image appears as the **File** selected and click the **OK** button.



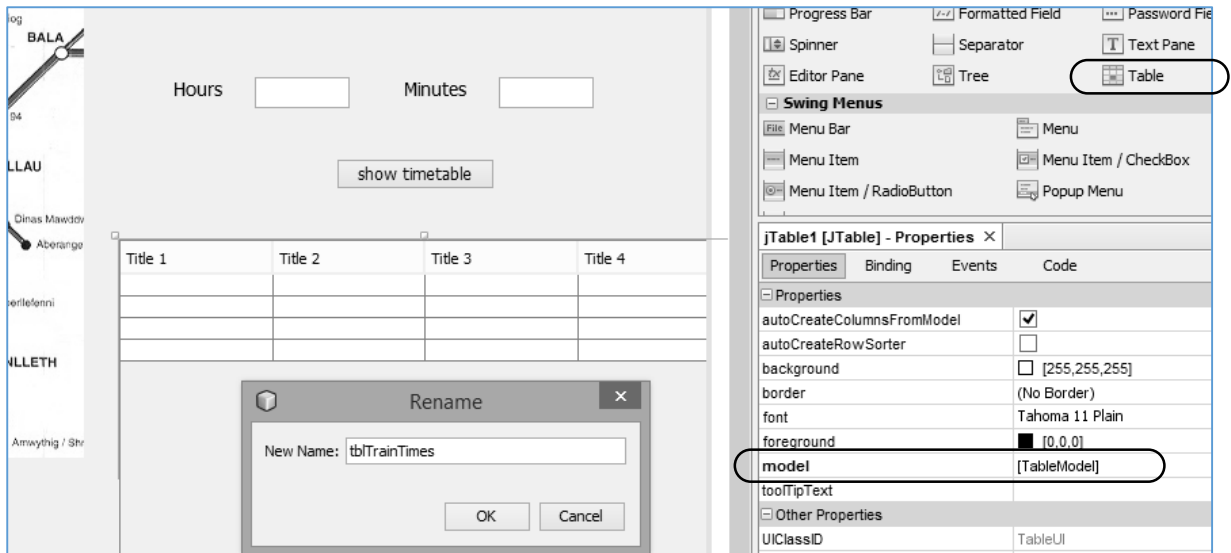
The image should appear on the form. Delete the caption for the label.

Add further labels '*Departure time from Machynlleth*', '*Hours*' and '*Minutes*', along with two *text fields* for entering the departure hours and minutes from the first station, *Machynlleth*. Rename the text fields as *txtHours* and *txtMinutes*. Also add a *button* with the caption "*show timetable*". Rename the button as *btnTimetable*:



Run the program and check that the form is displayed correctly. Close the program window and return to the NetBeans editing screen.

To display the timetable, it will be convenient to use a *table* component. Locate *Table* in the palette, and drag and drop this below the '*show timetable*' button. Right-click on the table and rename this as *tblTrainTimes*:

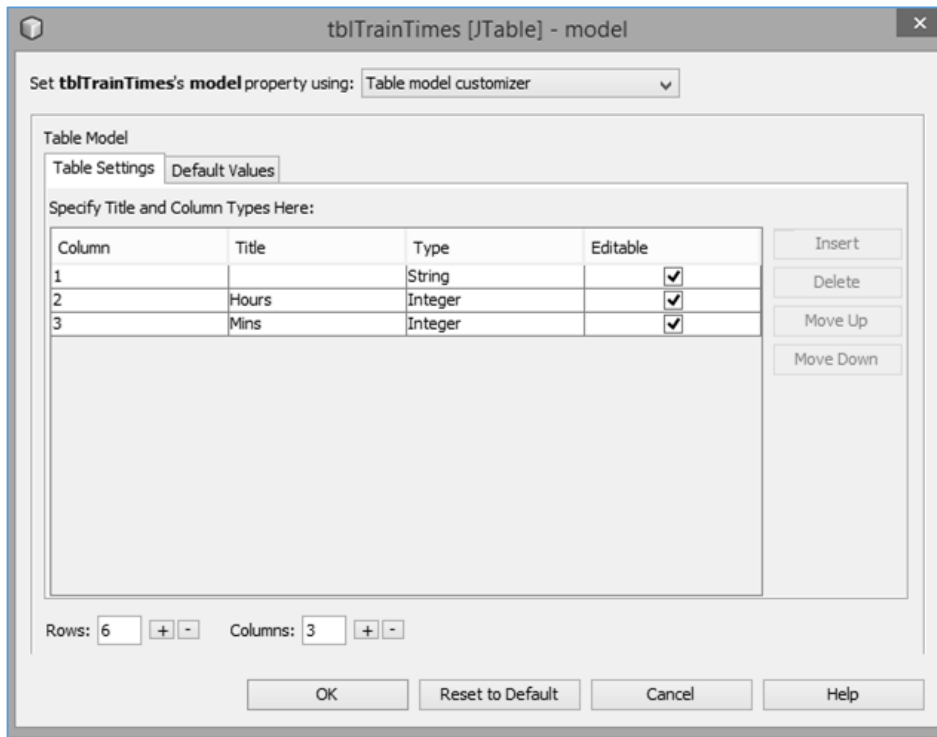


To set up the correct columns for the table, we will edit the *model* property. Click on *[TableModel]* to open an editing window.

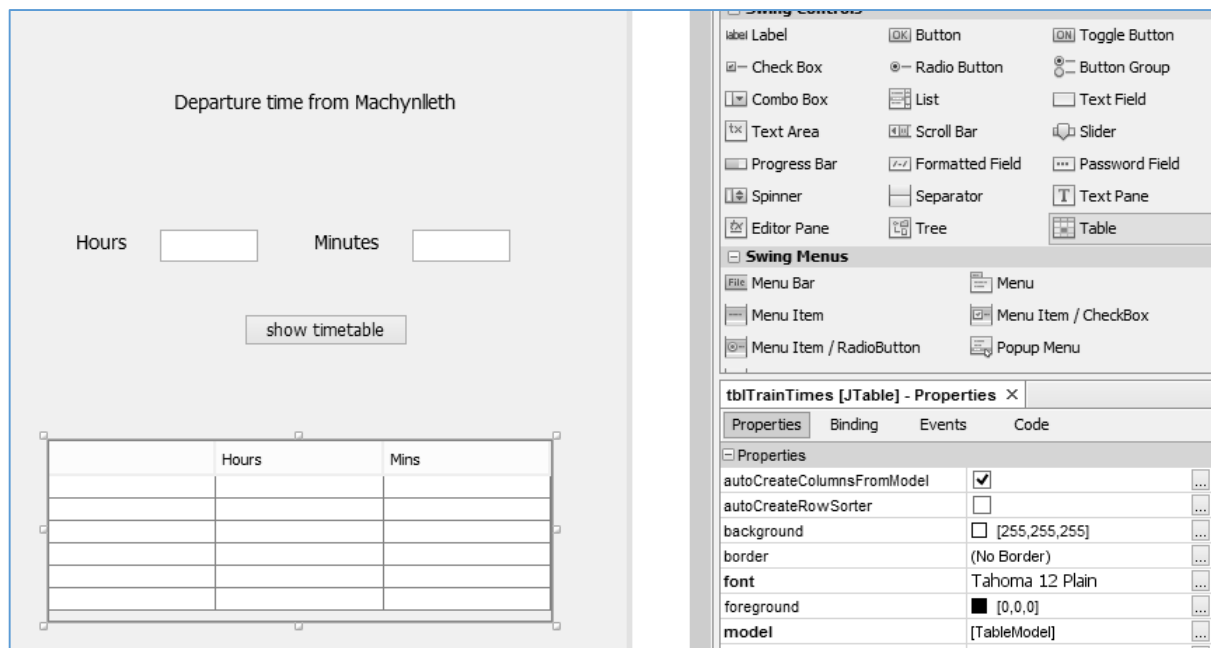
At the bottom of the editing window, set the number of **Rows** to **6**, and the number of **Columns** to **3**.

You should now specify the data for the table:

- The first column will display the names of the stations along the route, so contains **String** data. A heading is not needed.
- The second column contains the **hours** value of the train time. This will be an **integer** whole number. A heading "**Hours**" should be displayed above this column.
- The third column contains the **minutes** value of the train time. This will again be an **integer**. A heading "**Mins**" should be displayed:



Click the **OK** button to return to the editing screen. The table should now have the correct numbers of columns and rows, and should display headings for the **hours** and **minutes** columns:



Click the **Source** tab to open the program code window. We will now set up the list of stations in the table, and create an array holding the journey times between station stops.

At the start of the **timetable class**, we will add definitions for the **journey array**, and **integer variables** to hold the departure time in hours and minutes for the journey.

We go next to the **timetable() method**. This is the first method to run when the program is launched, so it is a good place to initialise the properties or variables which will be needed later in the program.

The first block of lines, such as:

```
tblTrainTimes.getModel().setValueAt("Barmouth",2,0);
```

will put the names of the stations into the table. The columns and rows are numbered, beginning at zero in the top left corner of the table. The station name '**Barmouth**' will be displayed on **row 2**, in **column 0**.

The second block of lines such as:

```
journey[1]=26;
```

will load the journey times into the correct elements of the **journey** array. In this case, the journey time for the first leg of the journey, from **Machynlleth** to **Tywyn**, is being set to **26 minutes**.

```
package timetablePackage;

import javax.swing.JOptionPane;

public class timetable extends javax.swing.JFrame {

    int[] journey=new int[6];
    int starthours, startminutes;

    public timetable() {
        initComponents();

        tblTrainTimes.getModel().setValueAt("Machynlleth",0,0);
        tblTrainTimes.getModel().setValueAt("Tywyn",1,0);
        tblTrainTimes.getModel().setValueAt("Barmouth",2,0);
        tblTrainTimes.getModel().setValueAt("Harlech",3,0);
        tblTrainTimes.getModel().setValueAt("Porthmadog",4,0);
        tblTrainTimes.getModel().setValueAt("Pwllheli",5,0);

        journey[1]=26;
        journey[2]=27;
        journey[3]=22;
        journey[4]=19;
        journey[5]=24;
    }
}
```

Run the program and check that the station names are displayed correctly in the table:

	Hours	Mins
Machynlleth		
Tywyn		
Barmouth		
Harlech		
Porthmadog		
Pwllheli		

Close the program window to return to the NetBeans editing screen. Click the Design tab to move to the form layout view. Double click the "**show timetable**" button to create a **method**.

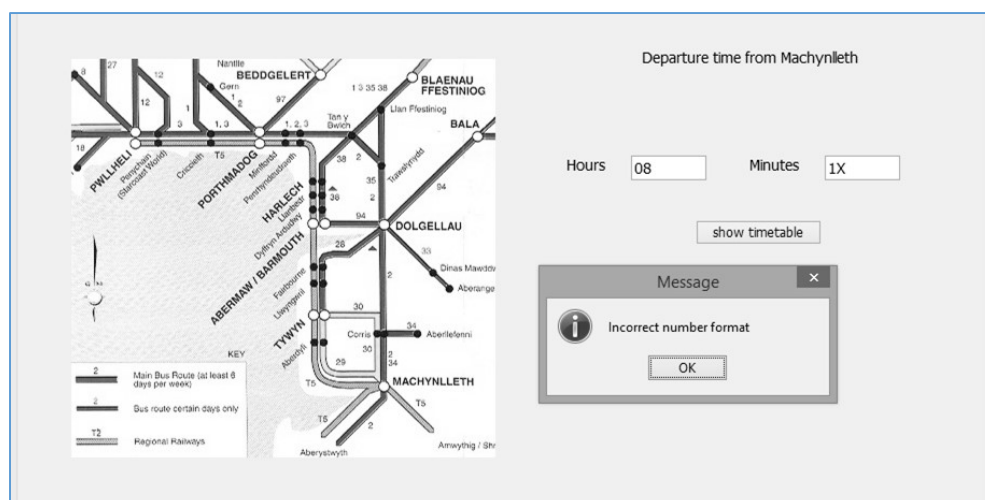
The first stage in producing the timetable is to obtain the departure time of the train service entered by the user. Add lines of program code to read the hour and minute values from the text fields. If either of the text fields has been left blank, then the corresponding variable will be set to zero.

We should also allow for an incorrect entry, so a **TRY...CATCH** block has been set up to display an error message box:

```
private void btnTimetableActionPerformed(java.awt.event.ActionEvent evt) {
    String s;
    int hours, minutes;
    try
    {
        if (txtMinutes.getText().equals(""))
        {
            startminutes=0;
        }
        else
        {
            s= txtMinutes.getText();
            startminutes= Integer.parseInt(s);
        }

        if (txtHours.getText().equals(""))
        {
            starthours=0;
        }
        else
        {
            s= txtHours.getText();
            starthours= Integer.parseInt(s);
        }
    }
    catch(NumberFormatException e)
    {
        JOptionPane.showMessageDialog(timetable.this,"Incorrect number format");
    }
}
```

Run the program. Enter correct values for departure hours and minutes, and check that these are accepted by the program. If a value is entered in an incorrect format, an error message should be displayed when the '**show timetable**' button is clicked:



Close the program and return to the NetBeans editing screen. We will add program code to the button click method.

- We will begin by carrying out *range checks* on the departure hours and minutes with the lines:

```
if (hours>=0 && hours<=23)...  
if (minutes>=0 && minutes<=59)...
```

The timetable should not be displayed if either of these values is outside the allowed range.

- If the hour and minute entries are valid, the program displays the departure time from the first station, Machynlleth, on the top line of the table

```
tblTrainTimes.getModel().setValueAt(hours,0,1);  
tblTrainTimes.getModel().setValueAt(minutes,0,2);
```

```
if (txtHours.getText().equals(""))  
{  
    starthours=0;  
}  
else  
{  
    s= txtHours.getText();  
    starthours= Integer.parseInt(s);  
}  
  
hours=starthours;  
minutes=startminutes;  
if (hours>=0 && hours<=23)  
{  
    if (minutes>=0 && minutes<=59)  
    {  
        tblTrainTimes.getModel().setValueAt(hours,0,1);  
        tblTrainTimes.getModel().setValueAt(minutes,0,2);  
    }  
}  
}
```

Run the program. Check that valid departure times are accepted by the program and displayed in the table, but hour or minute values which are out of range are not displayed:

The screenshot shows a Java Swing window with a light gray background. At the top, there are two text input fields: 'Hours' containing '12' and 'Minutes' containing '34'. Below these fields is a button labeled 'show timetable'. Underneath the button is a table with the following structure:

	Hours	Mins
Machynlleth	12	34
Tywyn		
Barmouth		
Harlech		
Porthmadog		
Pwllheli		

Close the program and return to the button click method on the source code screen.

We will now add a **loop** which operates five times to display the times for the remaining five stations. Each time around the loop, the program collects the journey time from the corresponding array element. This is added to the current minute value.

If the minute total now exceeds 59, the hour must be increased by one, and sixty minutes removed from the current minute total. For example: if a train departs at **9 hours 50 minutes**, a journey time of **26 minutes** will take the minute total to **76 minutes**, but the time should actually become **10 hours 16 minutes**:

```

if (hours>=0 && hours<=23)
{
    if (minutes>=0 && minutes<=59)
    {
        tblTrainTimes.getModel().setValueAt(hours,0,1);
        tblTrainTimes.getModel().setValueAt(minutes,0,2);

        for (int i=1; i<=5; i++)
        {
            minutes += journey[i];
            if (minutes>59)
            {
                minutes -=60;
                hours ++;
            }

            tblTrainTimes.getModel().setValueAt(hours,i,1);
            tblTrainTimes.getModel().setValueAt(minutes,i,2);
        }
    }
}

```

Notice the structure of the loop control line:

```
for ( int i = 1; i <= 5; i++)
```

This has three parts, separated by semi-colons. The first part:

```
int i = 1;
```

defines a **local variable** *i* which will be used as a **counter** while the loop is repeating. We are giving this variable an initial value of 1.

The next part:

```
i <= 5;
```

tells the computer to keep repeating the loop as long as the value of *i* is **less than or equal to 5**. After that, the loop will end.

The final section:

```
i++
```

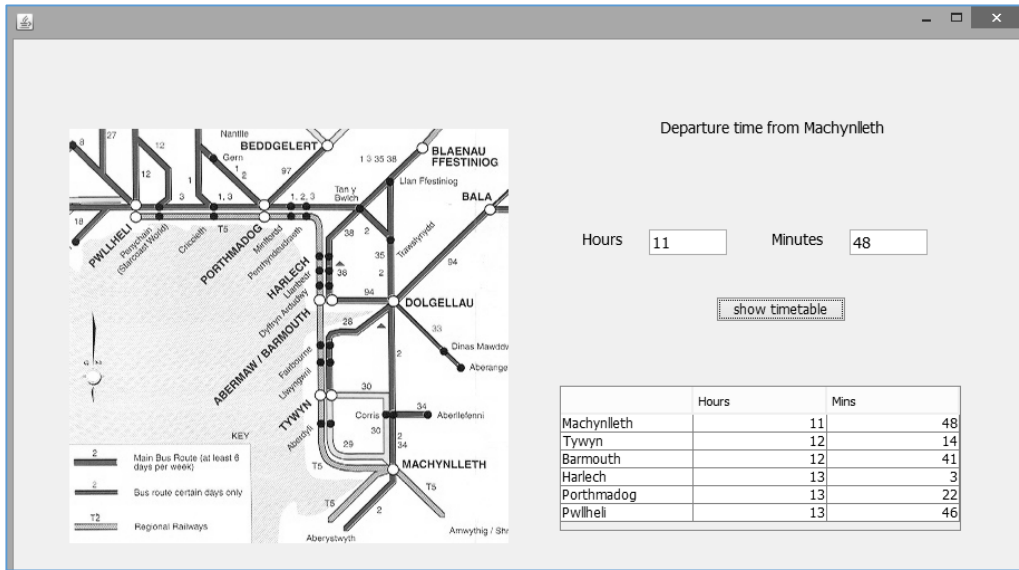
tells the computer to **add 1 to i** each time around the loop.

You will see in other programs that loop control structures in Java are very flexible. For example:

```
for ( int year = 1990; year <= 2050; year = year + 10)
```

would create year values from 1990 to 2050, increasing in steps of 10 years each time around the loop.

Run the program. Enter a variety of departure times and check that the correct times are displayed in the table for each of the stations. The hour should be incremented correctly where necessary during the journey.



You may discover that one small problem still remains: if a train departs shortly before midnight, arrival times after midnight will show an incorrect hour.

Close the program and return to the button click method. We will add lines of code to correct the hour value if this exceeds 23:

```

for (int i=1; i<=5; i++)
{
    minutes += journey[i];
    if (minutes>59)
    {
        minutes -=60;
        hours ++;
    }

    if (hours>23)
    {
        hours =0;
    }

    tblTrainTimes.getModel().setValueAt(hours,i,1);
    tblTrainTimes.getModel().setValueAt(minutes,i,2);
}
    
```

Run the completed program and check that journeys extending over midnight are timetabled correctly:

	Hours	Mins
Machynlleth	22	30
Tywyn	22	56
Barmouth	23	23
Harlech	23	45
Porthmadog	0	4
Pwllheli	0	28

The next program again involves the input and processing of data using arrays and loops:

A bus company carries out a survey of the bus services departing from a town centre. The objective is to compare the numbers of passengers travelling at different times of the day, and to identify heavily used services. A program is required to analyse the data collected. The program should input

- the number of buses surveyed

then for each bus:

- the departure time - hours
- the departure time - minutes
- the service number
- the number of passengers

The program should output:

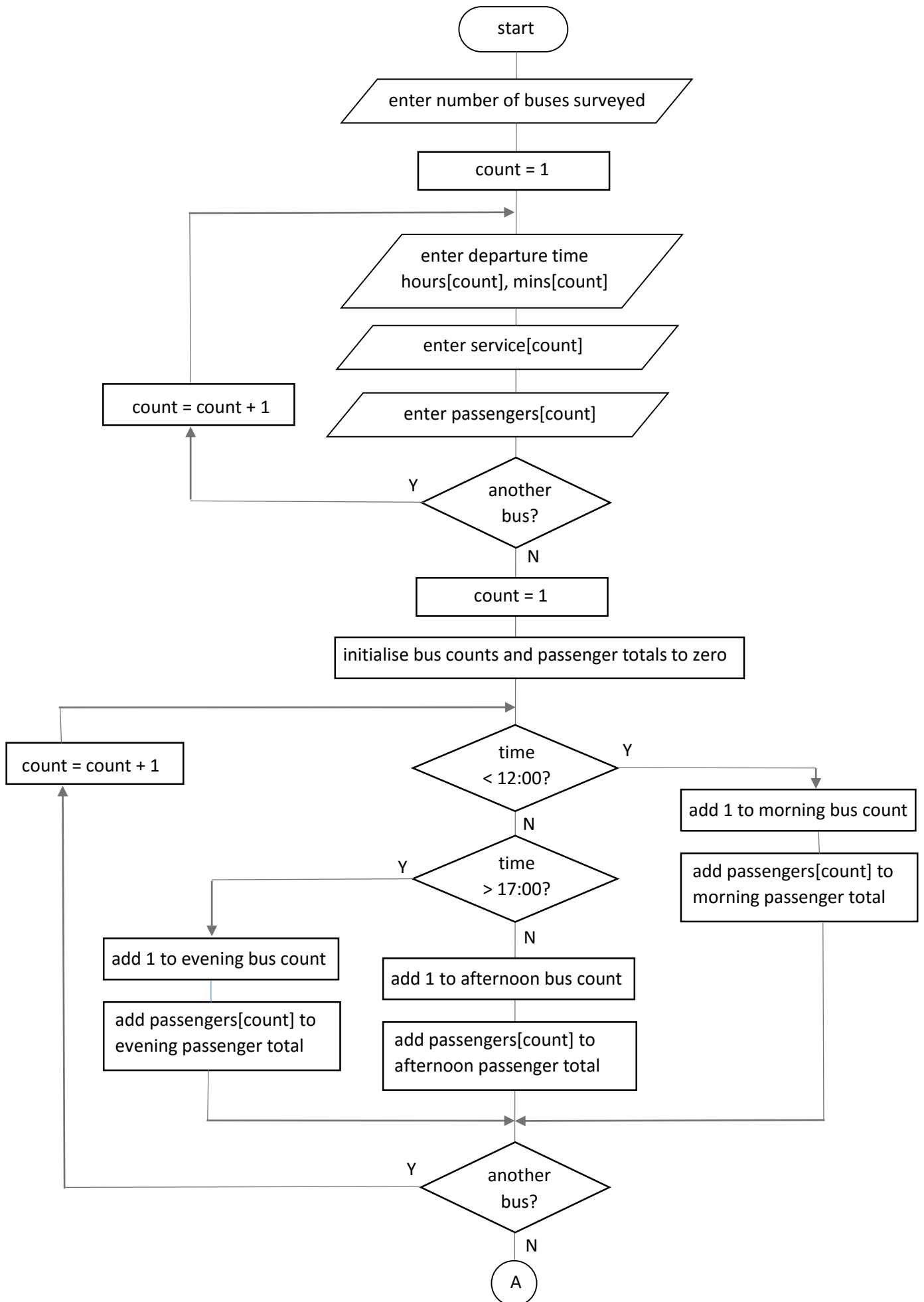
- the average numbers of passengers travelling during each part of the day:
 - Morning before 12:00 noon
 - Afternoon 12:00 noon – 5:00pm
 - Evening after 5:00pm
- a list all services with over 30 passengers, giving the departure time and service number.

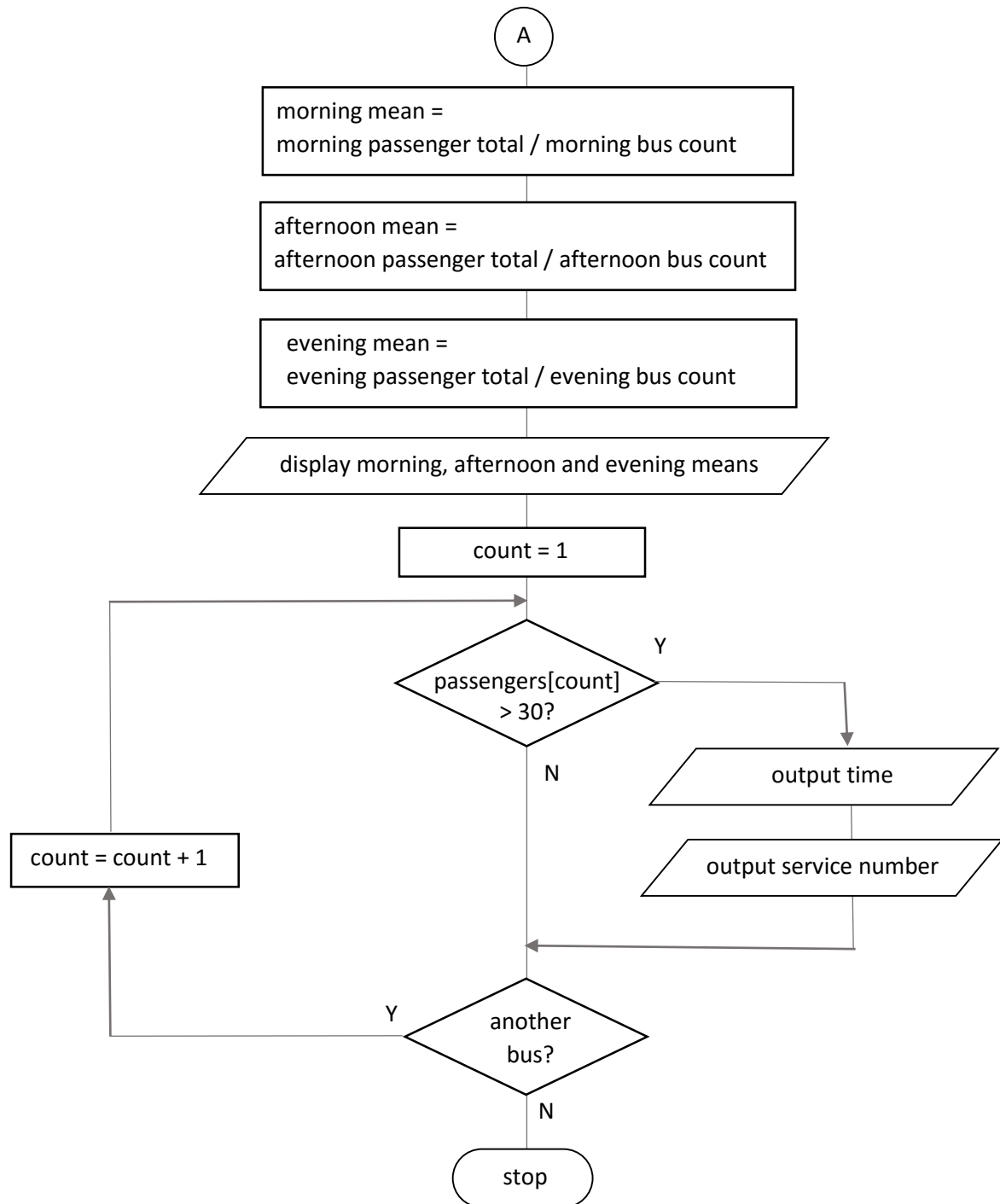
From the specification, we can see that four pieces of information need to be entered for each bus, two *integer* numbers for the departure time in *hours* and *minutes*, a *string* for the *service number* (which could possibly include a letter, such as 38B), and an *integer* to record the number of *passengers*. A convenient way to store this data will be in four *parallel arrays*:

9	11	13	16	...
hours[1]	hours [2]	hours [3]	hours [4]	...
20	15	45	0	...
mins[1]	mins [2]	mins [3]	mins [4]	...
38	S16	24X	38	...
service[1]	service[2]	service[3]	service[4]	...
38	26	19	35	...
pass[1]	pass[2]	pass[3]	pass[4]	...

Array elements with the same index number refer to the same event. For example, the elements with the *index value of 2* show *service S16* departing at *11:15* had *26 passengers*.

A design for the program is given on the next pages.





Close all projects, then set up a **New Project**. Give this the name **busSurvey**, and ensure that the **Create Main Class** option is not selected. Click **Finish** to return to the NetBeans editing screen.

Right-click on the **busSurvey** project, and select **New / JFrame Form**. Give the **Class Name** as **busSurvey**, and the **Package** as **busSurveyPackage**. Click **Finish** 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 editing screen.

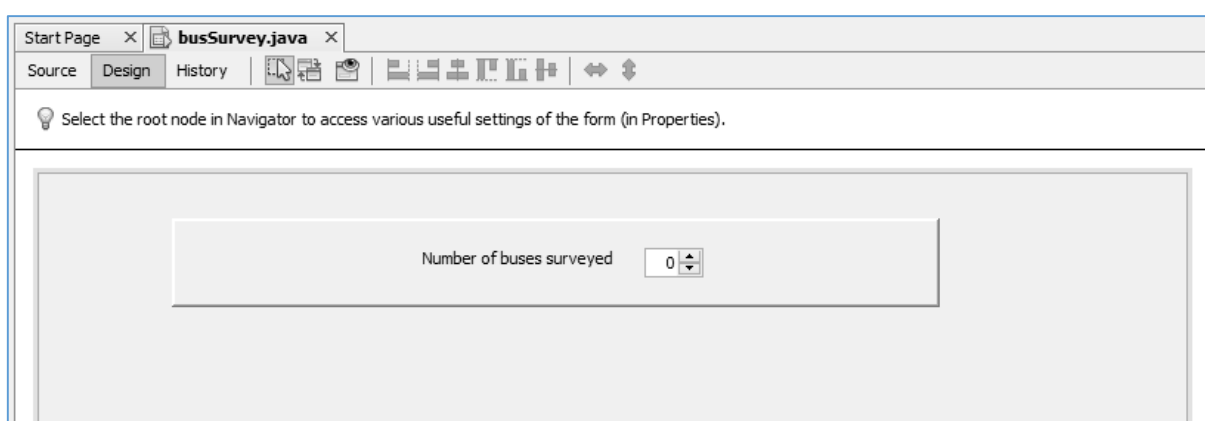
We will begin by setting up the arrays needed to store the survey data. Click the **Source** tab to open the program code page, then add the **arrays** and a **busCount** variable near the start of the program.

```
public class busSurvey extends javax.swing.JFrame {
    int[] hours=new int[100];
    int[] mins=new int[100];
    String[] service=new String[100];
    int[] passengers=new int[100];
    int busCount;

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

Return to the **Design** screen. Add a **Panel** component to the form, and set the **border** property to **BevelBorder**. Rename the panel as **pnlBusCount**. Right-click on the panel and select **Layout / Absolute Layout**.

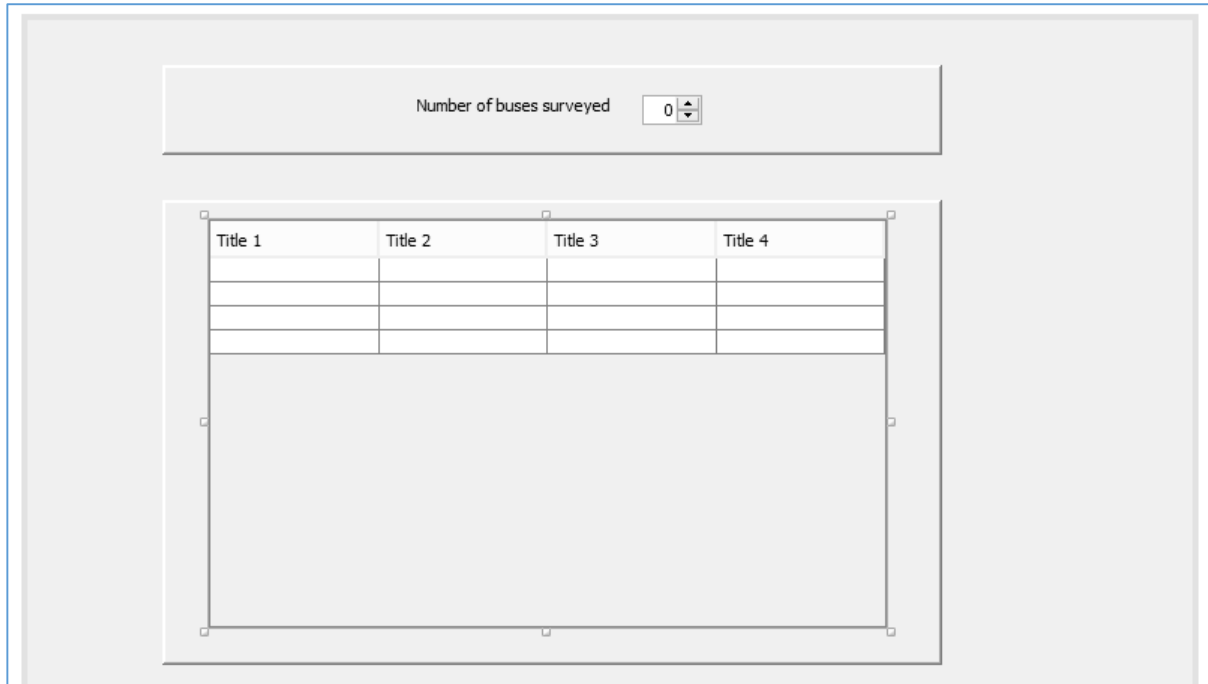
Add a **Label** and **Spinner** component to the panel. Set the text of the label to "**Number of buses surveyed**", and rename the spinner as **spinBusCount**. Widen the spinner to allow for multiple digits.



When the user has specified the number of buses surveyed, the input of survey results can begin.

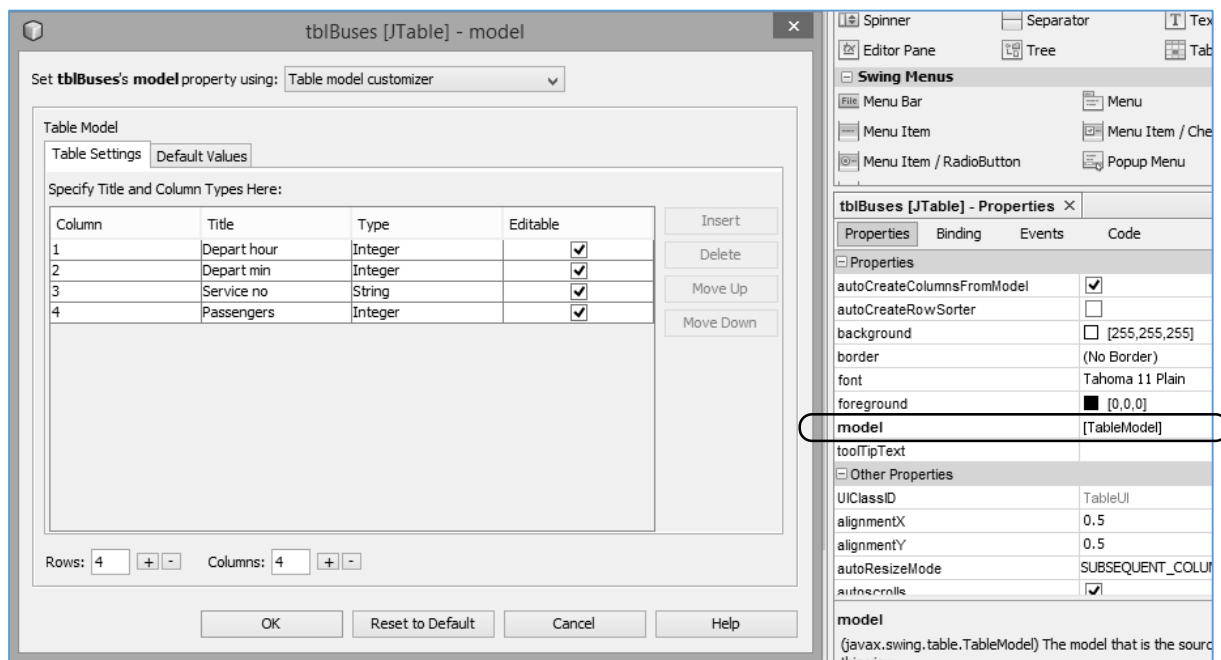
In previous programs we have entered data by means of text boxes, but there would be advantages in this case if a table similar to a spreadsheet was provided for data entry. This will allow the user to easily check the entries and correct any errors before the data is analysed.

Add another panel to the form, renaming this as *pnlTable*. Set the *border* property to *BevelBorder*. Drag and drop a *Table* component on the panel, renaming the table as *tblBuses*.



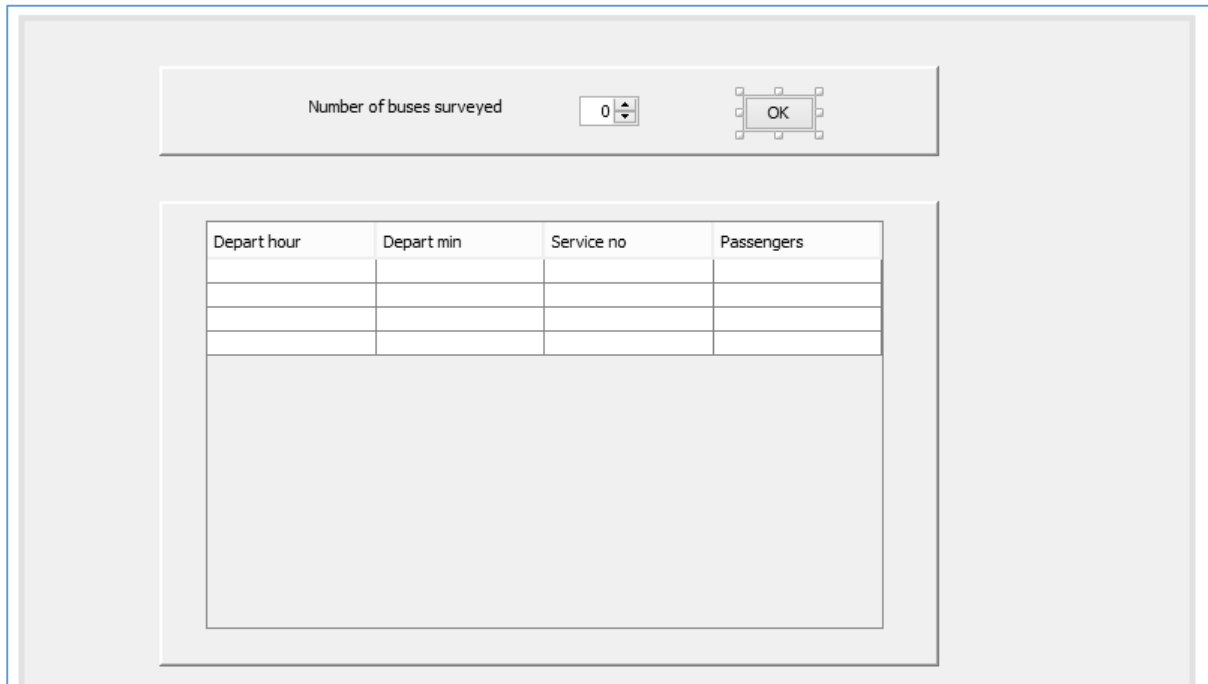
Locate the *model* property for the table, and click in the right hand column to open the editing window. Set the numbers of *rows* and *columns* to **4**. Add *titles* and *data types* for each of the columns:

Depart hour	Integer
Depart min	Integer
Service no	String
Passengers	Integer



Click the **OK** button to return to the design screen. Check that the column headings are displayed correctly in the table.

Add a **button** with the caption "**OK**" to the upper panel. Rename the button as **btnOK**:



Go to the Source code page. Add a line of code to the **busSurvey** method to make the panel containing the table **not visible** when the program first runs:

```
String[] service=new String[100];
int[] passengers=new int[100];
int busCount;

public busSurvey() {
    initComponents();

    pnlTable.setVisible(false);
}
```

Return to the **Design** screen to display the form, then double click the "**OK**" button to create a **method**. The user will click this button after entering the number of buses in the survey. It is then necessary to make the input table **visible**, and set up a corresponding number of blank rows for entering the survey results. To do this, we must access the **model** property of the table. Enter lines of program code to carry out these tasks:

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

    pnlTable.setVisible(true);
    DefaultTableModel model = (DefaultTableModel) tblBuses.getModel();
    busCount=(Integer) spinBusCount.getValue();
    model.setRowCount(busCount);
    tblBuses.setModel(model);
}
```

Before going further with the programming, scroll to the top of the program listing and add three code modules which will be needed when the program runs. These will allow us to input data into the table, and display of an error message box if necessary:

```
package busSurveyPackage;

import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellEditor;

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

Run the program. Enter different numbers of buses, using the spin component, and check that the correct numbers of blank lines are created in the table.

Enter some data into the table. You may notice that the table component has its own error checking procedure. If a value is entered in an incorrect format, for example: entering a letter in the Passengers column, then a red outline will appear around the grid cell. It will not be possible to enter further values until the error is corrected.

Depart hour	Depart min	Service no	Passengers
8	0	S6	39
10	30	W102	16
12	25	Z32	23

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

Add another **panel** below the table, renaming this as **pnlResults**. Place a **text area** on the panel, with the name **txtResults**. Complete the form design by adding a **button** to the table panel with the caption "**Analyse data**". Rename the button as **btnResults**.

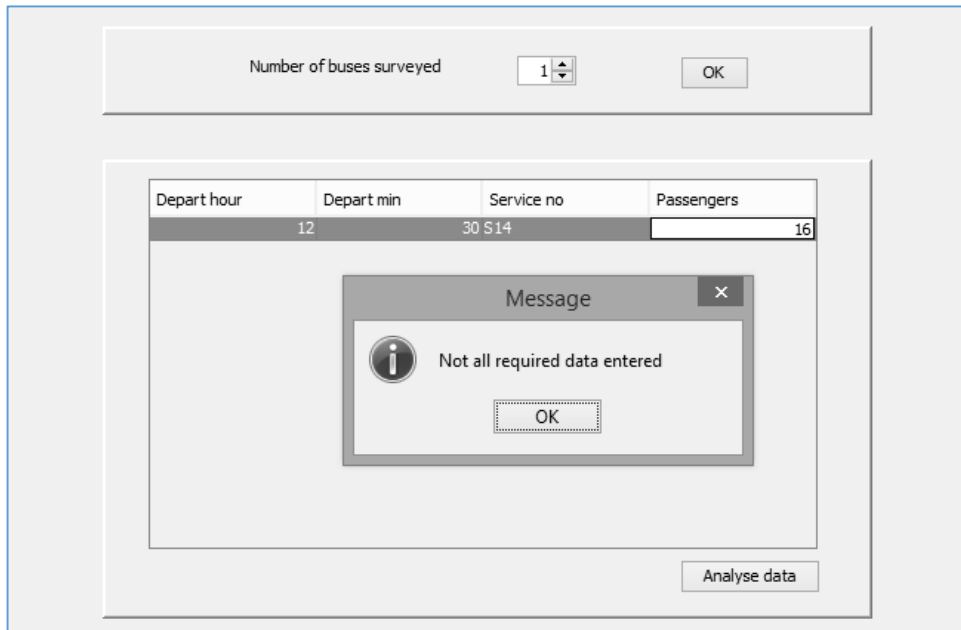
Depart hour	Depart min	Service no	Passengers

Analyse data

Double click the "**Analyse data**" to create a **method**. We will begin by collecting the input data from the table using a loop structure, and storing the values in the **parallel arrays**. The loop can be enclosed in a **TRY ... CATCH** structure to give an error message if data values are missing.

```
private void btnResultsActionPerformed(java.awt.event.ActionEvent evt) {
    pnlResults.setVisible(true);
    try
    {
        for (int i=0;i<busCount; i++)
        {
            hours[i]= (Integer) tblBuses.getModel().getValueAt(i,0);
            mins[i]= (Integer) tblBuses.getModel().getValueAt(i,1);
            service[i]= (String) tblBuses.getModel().getValueAt(i,2);
            passengers[i]= (Integer) tblBuses.getModel().getValueAt(i,3);
        }
    }
    catch(NullPointerException e2)
    {
        JOptionPane.showMessageDialog(busSurvey.this,
            "Not all required data entered");
    }
}
```

Run the program and test the data entry.



You may discover that the program does not behave as expected. If a set of data is entered correctly, the "**missing data**" error message may still appear. This is because data items are not recorded by the table until the user presses the **Enter** key or clicks on another grid cell. We can attend to this problem now.

Close the program window and return to the source code page. Add lines of program code which will force the table to stop editing and accept the current cell value when the "**Analyse data**" button is clicked.

```
private void btnResultsActionPerformed(java.awt.event.ActionEvent evt) {
    TableCellEditor editor = tblBuses.getCellEditor();
    if (editor != null)
    {
        editor.stopCellEditing();
    }

    pnlResults.setVisible(true);
    try
    {
        for (int i=0;i<busCount; i++)
        {
            hours[i]= (Integer) tblBuses.getModel().getValueAt(i,0);
        }
    }
}
```

Run the program and check that the table input now works correctly. Close the program window and return to the source code page.

A requirement for the program is to display the mean number of passengers using the buses at different times of day. We will begin by calculating the mean number of morning passengers per bus.

We initialise the numbers of morning buses and passengers to zero. A loop is then used to check each bus in turn. If the departure time is before 12:00 noon, the number of morning buses is increased by one, and the number of passengers is added to the morning passenger total.

When the loop ends, the mean is calculated and displayed in the text area:

```
try
{
    for (int i=0;i<busCount; i++)
    {
        hours[i]= (Integer) tblBuses.getModel().getValueAt(i,0);
        mins[i]= (Integer) tblBuses.getModel().getValueAt(i,1);
        service[i]= (String) tblBuses.getModel().getValueAt(i,2);
        passengers[i]= (Integer) tblBuses.getModel().getValueAt(i,3);
    }

    double morningPassengers=0;
    double morningBuses=0;
    double morningMean=0;
    for (int i=0;i<busCount; i++)
    {
        if (hours[i]<12)
        {
            morningBuses++;
            morningPassengers += passengers[i];
        }
    }
    if (morningBuses>0)
    {
        morningMean = morningPassengers/morningBuses;
    }
    else
    {
        morningMean = 0;
    }
    String s="";
    s+="Average passengers: \n";
    s+= "Morning      "+String.format("%.1f",morningMean)+"\n";
    txtResults.setText(s);
}
catch(NullPointerException e)
```

Run the program. Add some morning bus services and click the button to analyse the data. Check that the average number of passengers for each morning bus is shown correctly:

Depart hour	Depart min	Service no	Passengers
9	30	S12	38
11	15	D19	17

Analyse data

Average passengers:
Morning 27.5

Close the program and return to the source code page. Add similar lines of code to calculate the mean numbers of passengers for afternoon and evening services:

```
double morningPassengers=0;
double morningBuses=0;
double morningMean=0;

double afternoonPassengers=0;
double afternoonBuses=0;
double afternoonMean=0;
double eveningPassengers=0;
double eveningBuses=0;
double eveningMean=0;

for (int i=0;i<busCount; i++)
{
    if (hours[i]<12)
    {
        morningBuses++;
        morningPassengers += passengers[i];
    }

    else
    {
        if (hours[i]>=17 )
        {
            eveningBuses++;
            eveningPassengers += passengers[i];
        }
        else
        {
            afternoonBuses++;
            afternoonPassengers += passengers[i];
        }
    }
}

if (morningBuses>0)
{
    morningMean = morningPassengers/morningBuses;
}
else
{
    morningMean = 0;
}

if (afternoonBuses>0)
{
    afternoonMean = afternoonPassengers/afternoonBuses;
}
else
{
    afternoonMean = 0;
}
```

```

if (eveningBuses>0)
{
    eveningMean = eveningPassengers/eveningBuses;
}
else
{
    eveningMean = 0;
}

String s="";
s+="Average passengers: \n";
s+= "Morning      "+String.format("%.1f",morningMean)+"\n";

s+= "Afternoon    "+String.format("%.1f",afternoonMean)+"\n";
s+= "Evening      "+String.format("%.1f",eveningMean)+"\n";

txtResults.setText(s);
}
catch(NullPointerException e)
{

```

Run the program. Enter buses for various times during the day and check that the mean numbers of passengers are displayed correctly:

The screenshot shows a Java Swing application window with the following components:

- A text field labeled "Number of buses surveyed" with the value "9" and an "OK" button.
- A table with the following data:

Depart hour	Depart min	Service no	Passengers
7	40	S11	28
8	30	D12	39
10	25	S14	41
13	6	D9	28
14	45	X44	16
16	20	S14	38
19	5	D12	22
20	16	S11	11
22	0	X14	19
- An "Analyse data" button.
- A text area displaying the output:

```

Average passengers:
Morning      36.0
Afternoon    27.3
Evening      17.3

```


Close the program and return to the source code page.

The final program requirement is to list all bus services with more than 30 passengers. Do this by adding another loop to the button click method. When buses with over 30 passengers are found, the program collects the hour and minute values then assembles these into a time format, separated by a colon. If a single digit number is found for the hour or minute value, then a zero is added to produce a standard format. For example: 9 hours 5 minutes is displayed as 09:05.

```

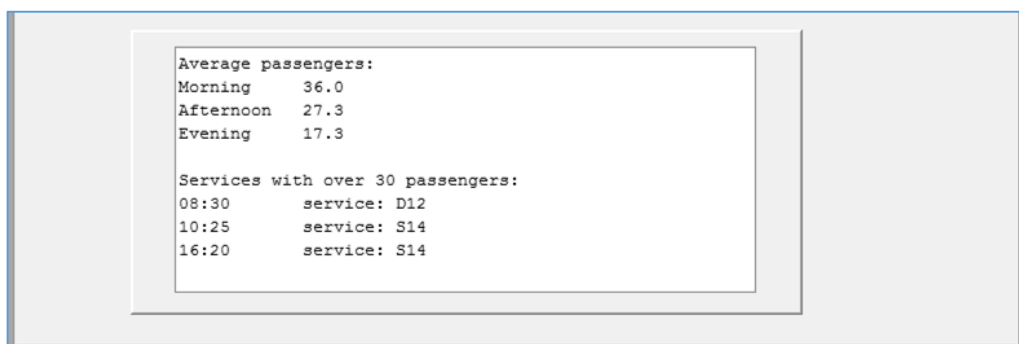
s+="Average passengers: \n";
s+= "Morning      "+String.format("%.1f",morningMean)+"\n";
s+= "Afternoon    "+String.format("%.1f",afternoonMean)+"\n";
s+= "Evening      "+String.format("%.1f",eveningMean)+"\n";

s+= "\nServices with over 30 passengers: \n";
int t;
String time;
for (int i=0;i<busCount; i++)
{
    if (passengers[i]>30)
    {
        t=hours[i];
        time=Integer.toString(t);
        if (t<10)
        {
            s += "0";
        }
        s+= time+":";
        t=mins[i];
        time=Integer.toString(t);
        if (t<10)
        {
            s += "0";
        }
        s+= time+"      service: "+service[i];
        s += "\n";
    }
}

txtResults.setText(s);
}
catch(NullPointerException e)
{

```

Run the program. Enter a set of test data and check that all bus services with more than 30 passengers are listed.



```

Average passengers:
Morning      36.0
Afternoon    27.3
Evening      17.3

Services with over 30 passengers:
08:30      service: D12
10:25      service: S14
16:20      service: S14

```