8 Sorting

Most data processing applications require data to be sorted in some way, perhaps into ***alphabetical order*** of customer surnames or ***numerical order*** of product identification codes. In this chapter we will examine a technique for sorting data using ***arrays***.

We begin with an application which requires text values to be sorted alphabetically:

> A college is compiling a list of the courses which it offers to students. After entering the course titles, the program should sort and display the list in alphabetical order.

Begin the project in the standard way. Close all previous projects, then set up a ***New Project***. Give this the name ***subjects***, and ensure that the ***Create Main Class*** option is not selected.

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

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 ***Design*** screen. Add a ***Table*** component to the form and rename this as ***tblCourse***.

Title 1	Title 2	Title 3	Title 4

Select the ***Table*** and go to the ***Properties*** window. Locate the model property and click in the right column to open the Table Settings window.

Set the number of ***Rows*** to **12** and ***Columns*** to **1**. Give the column title as "***Course***", and the data type as ***String***.

Click the ***OK*** button to close the table editing window. Check that the table is displayed correctly. Add a ***button*** below the table, giving this the caption "***Sort***" and the name ***btnSort***.

Use the ***Source*** tab to move to the program code view. Add a Java module '***TableCellEditor***'. Define an array '***course***' which will be used to hold the course data entered in the table, ready for sorting.

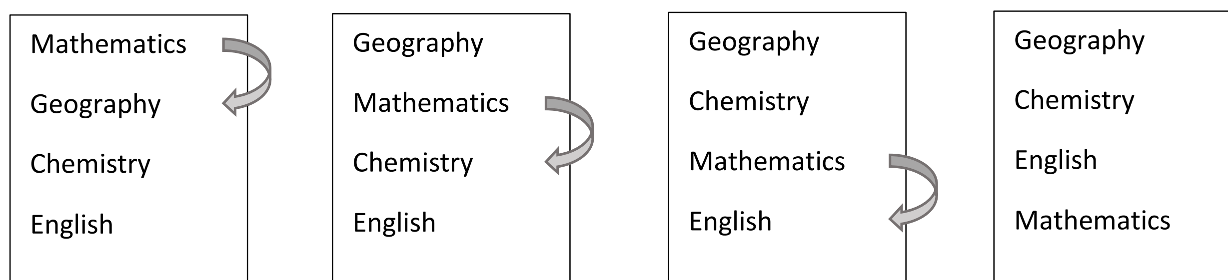
```
package subjectsPackage;

import javax.swing.table.TableCellEditor;

public class subjects extends javax.swing.JFrame {

    public static String[] course=new String[12];

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

Use the ***Design*** tab to return to the form layout view. Double click the "***Sort***" button to create a method.

We begin by stopping the table editor, to ensure that all text entries will be available for processing. The next step is to collect the data values from the table and insert these into the ***course*** array. We do not know in advance how many courses will be entered by the user, so a loop operates for each row of the table until a blank row is reached. The number of data values found in the table is recorded by the variable '***count***'.

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

    TableCellEditor editor = tblCourse.getCellEditor();
    if (editor != null)
    {
         editor.stopCellEditing();
    }
    int count=0;
    while (tblCourse.getModel().getValueAt(count,0)!=null)
    {
        course[count]= tblCourse.getModel().getValueAt(count,0).toString();
        count++;
        if (count==12)
        {
           break;
        }
    }

}
```

The ***course*** data is now in the array, and can be sorted using a method called a ***Bubble Sort***. This compares each pair of entries in the list in turn, and swaps the entries if they are not correct alphabetical order. For example:

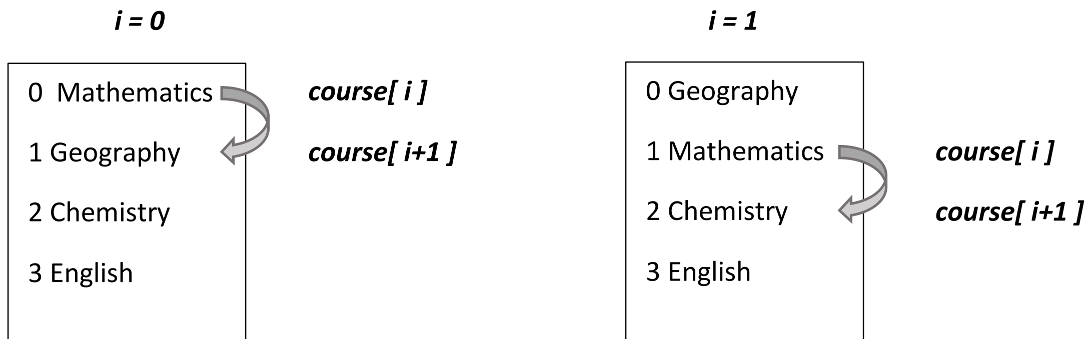


After the first pass through the data, the entries are not yet sorted correctly. However, the course names which are earlier in the alphabet have moved upwards in the list, whilst those later in the alphabet have moved down. The process can be repeated until the data is fully sorted.

Each pass through the data can be carried out with a ***loop***, using a ***loop counter variable i***:

for (int i = 0; i < count - 1; i++)

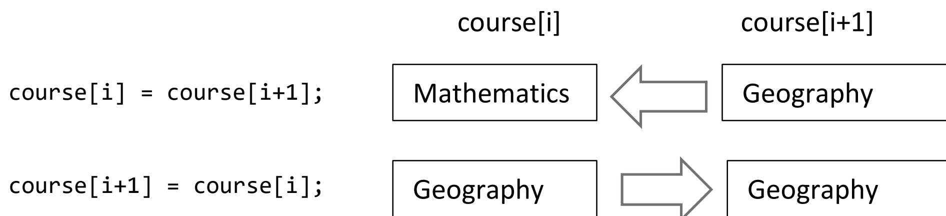
Each time, we are comparing the course at ***position i*** with the course below this at ***position (i+1)***



We must exchange the course names if these are not currently in the correct alphabetical order. At first glance, it might seem possible to do this by means of the lines:

```
course[i] = course[i+1];
course[i+1] = course[i];
```

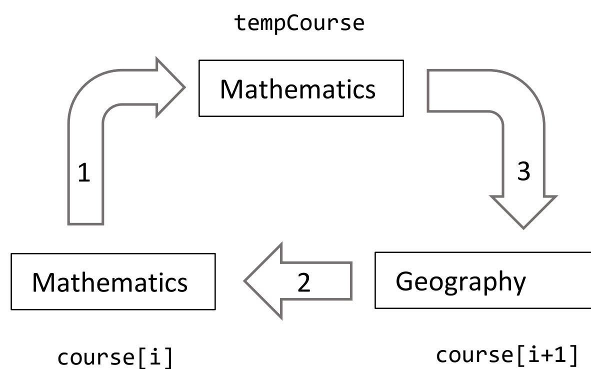
However, this would result in the two array elements containing the ***same*** data value:



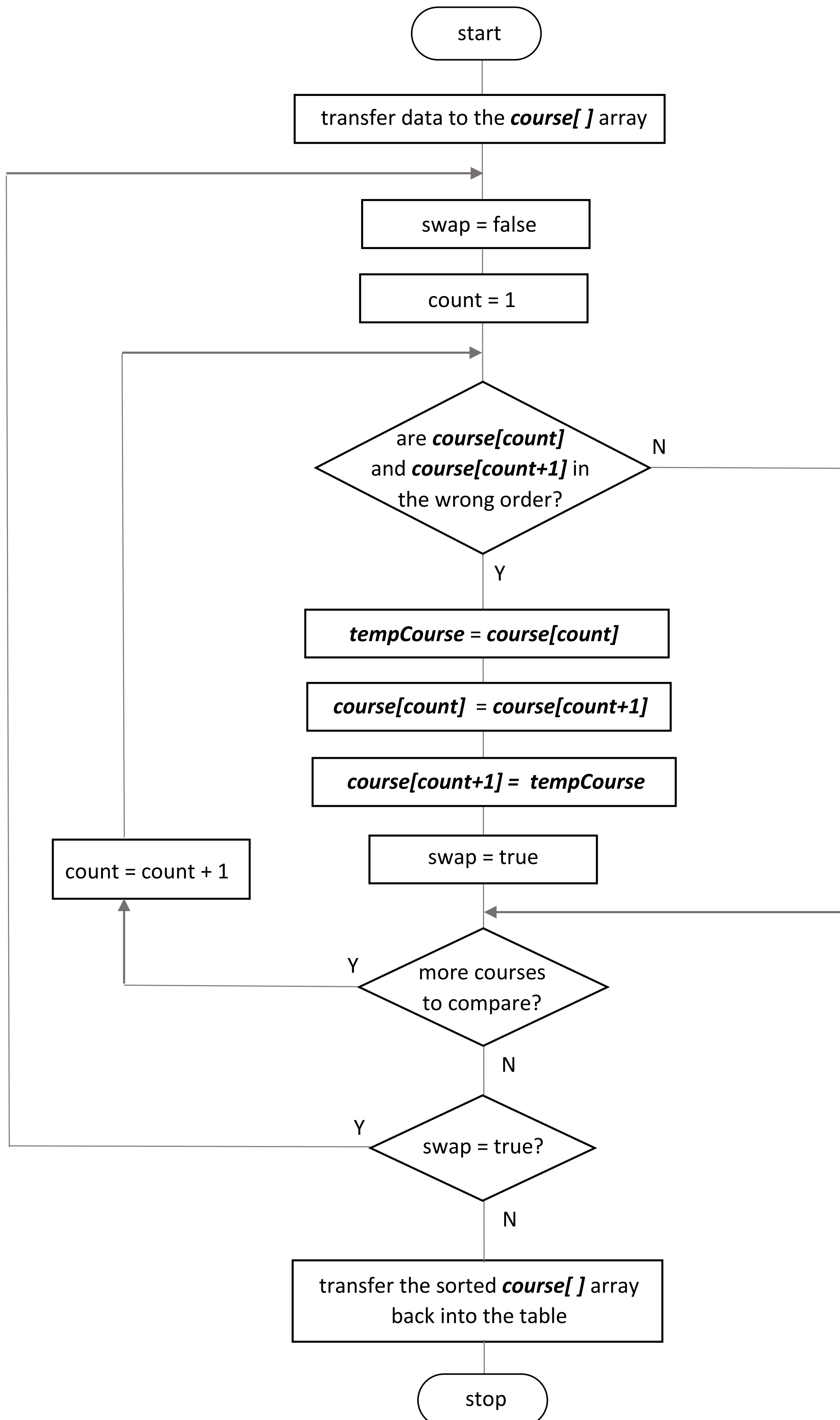
We can avoid this problem by introducing a temporary storage location:

```
tempCourse=course[i];
course[i]=course[i+1];
course[i+1]=tempCourse;
```

This technique, known as a ***triangular exchange***, allows the data items to be processed correctly:



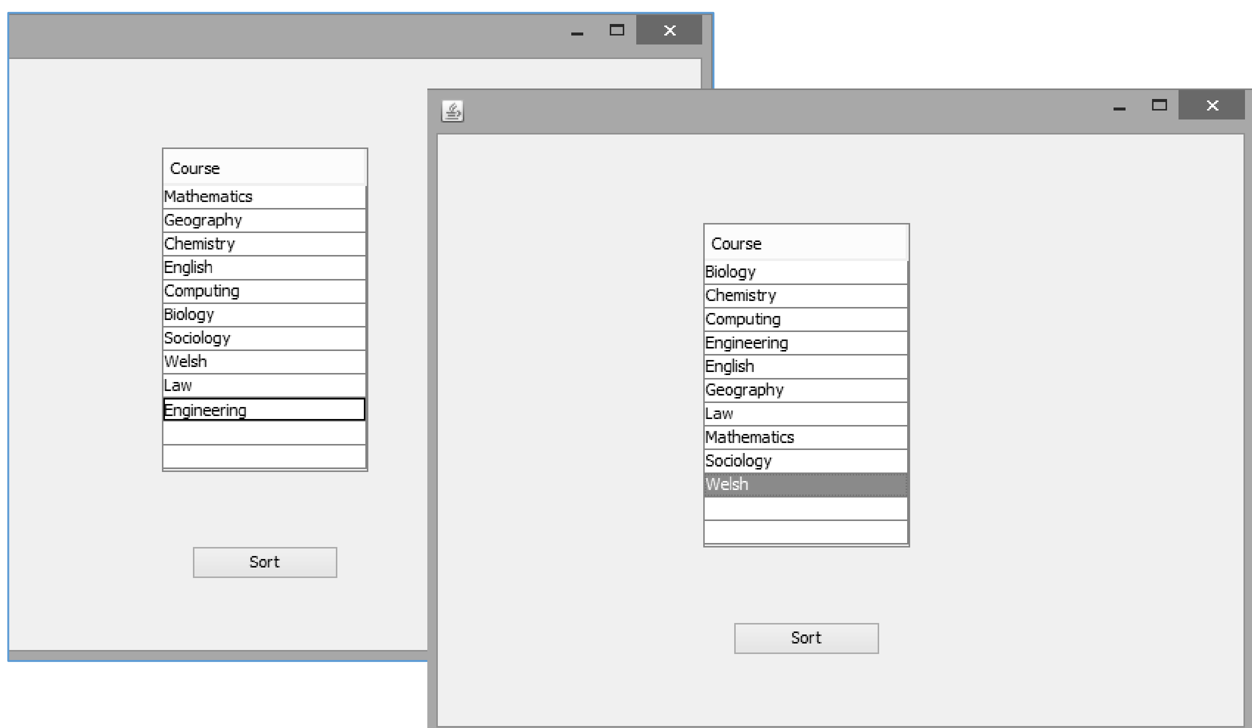
A flow chart for the sort process is given below. Notice how we use a ***Boolean*** variable '***swap***' to identify when the sorting is completed. The program continues to carry out passes through the ***course*** array until a loop can be completed without the need for any swapping of data items.



We will now implement the algorithm shown in the flowchart. Add lines of program code to the button click method:

```
    while (tblCourse.getModel().getValueAt(count,0)!=null)
    {
          course[count]= tblCourse.getModel().getValueAt(count,0).toString();
          count++;
          if (count==12)
          {
             break;
          }
    }
    Boolean swap=true;
    String tempCourse;
    while (swap==true)
    {
        swap=false;
        for (int i=0; i<count-1;i++)
        {
            if (course[i].compareTo(course[i+1])>0)
            {
                tempCourse=course[i];
                course[i]=course[i+1];
                course[i+1]=tempCourse;
                swap=true;
            }
        }
    }
    for(int i=0;i<count;i++)
    {
        tblCourse.getModel().setValueAt(course[i],i,0);
    }
}
```

Run the program. Enter a series of course titles, then click the "***Sort***" button. Check that the courses now appear in correct alphabetical order.



For the next project we will create a more substantial data processing application combining input, storage, sorting and display of data records.

> The organisers of a marathon require a program for recording and processing the results. Up to 50 runners may be taking part.
>
> - Before the race, the surnames and forenames of runners will be entered and stored on disc. It should be possible to reload this data, add the names of further competitors and re-save the file.
> - After the race, the times taken by runners will be entered. Times are to be recorded in hours and minutes. Most runners are expected to take between 3 and 5 hours to complete the course.
> - It is required to sort the results in two ways:
>
>   (1) alphabetical order of competitors
>
>   (2) time taken for the run

Begin the project in the standard way. Close all previous projects, then set up a ***New Project***. Give this the name ***marathon***, and ensure that the ***Create Main Class*** option is not selected.

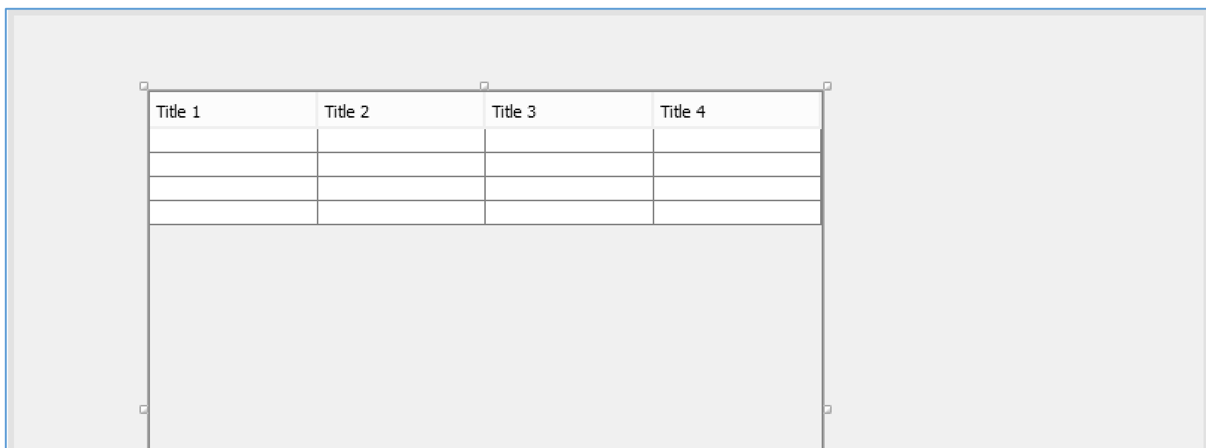
Return to the NetBeans editing page. Right-click on the ***subjects*** project, and select ***New*** / ***JFrame Form***. Give the ***Class Name*** as ***marathon***, and the ***Package*** as ***marathonPackage***:

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.

Close the program and return to the ***Design*** screen. Add a ***Table*** component to the form and rename this as ***tblResults***.



Select the ***Table*** and go to the ***Properties*** window. Locate the model property and click in the right column to open the Table Settings window.

Set the number of ***Rows*** to **50** and ***Columns*** to 5. ***Column 1*** will be a sequence number generated by the program. Leave the ***Title*** blank, set the field Type to ***Integer*** and remove the '***Editable***' tick. For the remaining columns, give the ***Titles*** and ***Data types*** as:

Surname	***String***
Forename	***String***
Hours	***Integer***
Minutes	***Integer***

tblResults [JTable] - model

Set **tblResults**'s **model** property using: Table model customizer

Table Model

Table Settings | Default Values

Specify Title and Column Types Here:

Column	Title	Type	Editable
1		Integer	☐
2	Surname	String	☑
3	Forename	String	☑
4	Hours	Integer	☑
5	Minutes	Integer	☑

Insert | Delete | Move Up | Move Down

Rows: 50 + - Columns: 5 + -

OK | Reset to Default | Cancel | Help

Click the ***OK*** button to return to the form design screen. Check that the table headings are displayed correctly. The table will display a vertical scroll bar to allow for the 50 rows of data.

Add buttons with the captions "***Save file***" and "***Load file***". Rename these as ***btnSave*** and ***btnLoad***.

	Surname	Forename	Hours	Minutes

Save file

Load file

Click the ***Source*** tab to move to the program code screen. We will begin by including the Java modules needed for saving and loading data, handling data errors and editing the table data.

We will set up a filename "***results.dat***" for storing the table data.

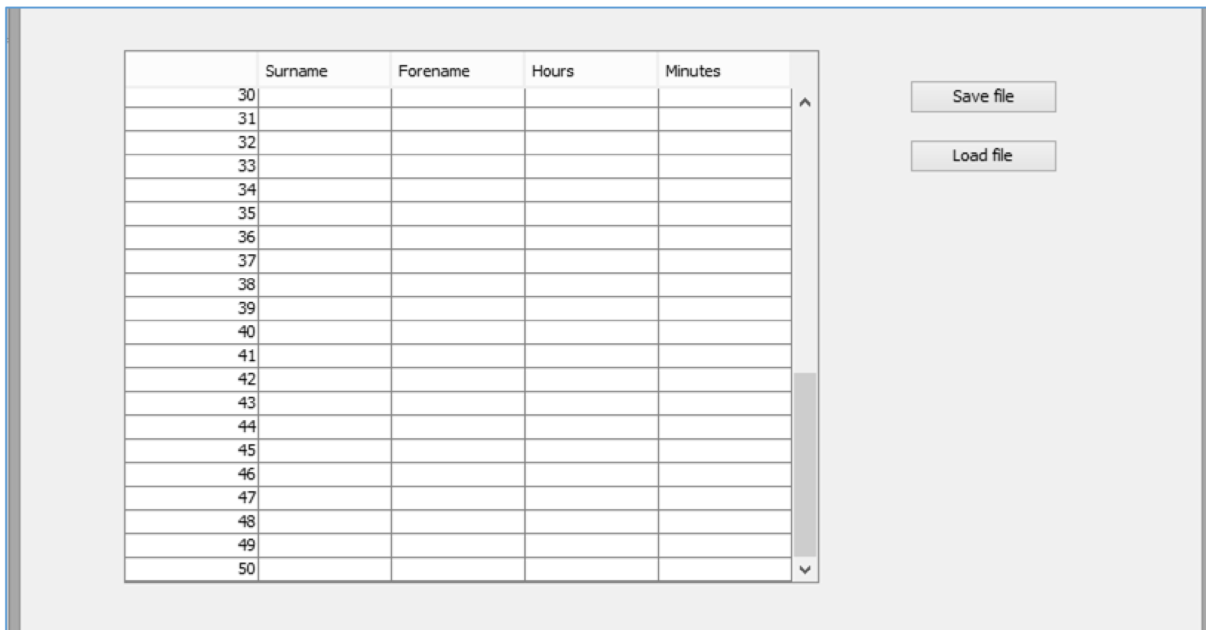
We will then add a series of variable definitions. Notice that ***arrays*** are being set up to hold the surnames, forenames and race times for the runners. We will also need temporary variables for the ***triangular exchange*** of data during sorting.

```
package marathonPackage;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import javax.swing.JOptionPane;
import javax.swing.table.TableCellEditor;

public class marathon extends javax.swing.JFrame {

  public static String filename="results.dat";
  public static String[] surname=new String[50];
  public static String[] forename=new String[50];
  public static int[] hours=new int[50];
  public static int[] minutes=new int[50];
  public static String tempSurname;
  public static String tempForename;
  public static int tempHours;
  public static int tempMinutes;

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

It will be convenient to provide a method to clear the table entries and set up the sequence numbers in the first column. We will call this method from ***marathon( )***, which is the first method to run when the program first begins.

```
public marathon() {
    initComponents();

    clearTable();

}

private void clearTable()
{
    for (int i=0;i<50;i++)
    {
        tblResults.getModel().setValueAt(i+1,i,0);
        tblResults.getModel().setValueAt("",i,1);
        tblResults.getModel().setValueAt("",i,2);
        tblResults.getModel().setValueAt("",i,3);
        tblResults.getModel().setValueAt("",i,4);
    }

}
```

Run the program. Check that the table shows rows numbered from 1 to 50:



Close the program and return to the NetBeans editor. Use the Design tab to move to the form layout view. Double click the "***Save file***" button to create a method.

The next step is to add code to the method to stop the table editor, to ensure that all data items are available for processing.

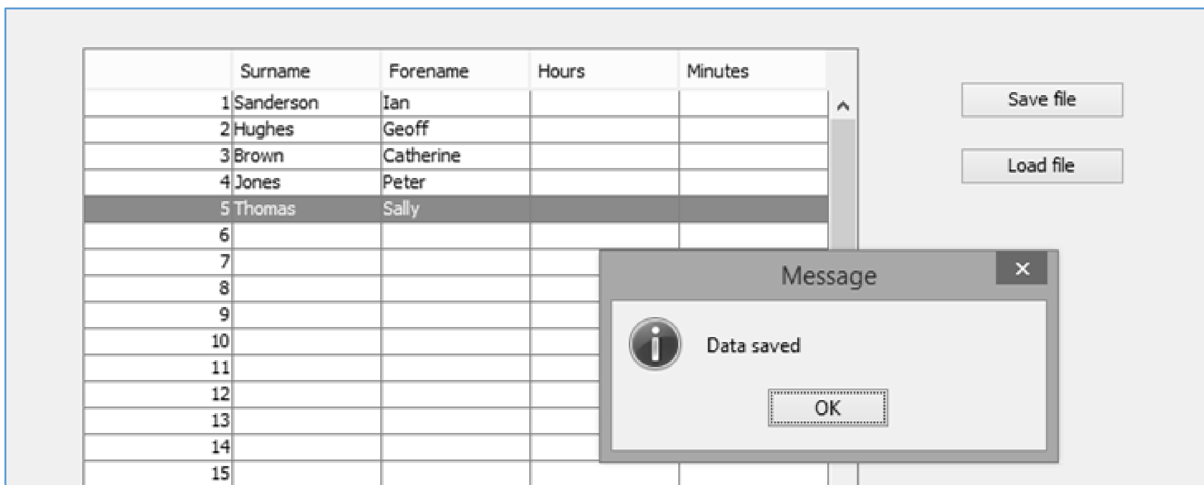
We will save the data as a text file with variable length records. We will create a ***TRY ... CATCH*** structure to handle errors, and add lines to open and close the ***results.dat*** file.

```
private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {
    TableCellEditor editor = tblResults.getCellEditor();
    if (editor != null)
    {
         editor.stopCellEditing();
    }
    try
    {
        FileWriter w = new FileWriter(filename);
        BufferedWriter writer = new BufferedWriter(w);

        writer.close();
        JOptionPane.showMessageDialog(marathon.this, "Data saved");
    }
    catch (IOException e)
    {
        JOptionPane.showMessageDialog(marathon.this, "File error");
    }
}
```

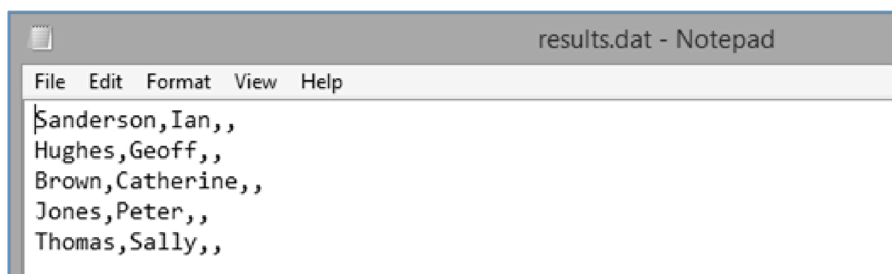
We will add a loop to check each row of the table. If a surname is found on the current row, the data values for this competitor will be collected, compiled into a record and saved into the file.

```
try
{
    FileWriter w = new FileWriter(filename);
    BufferedWriter writer = new BufferedWriter(w);

    for(int i=0;i<50;i++)
    {
        String s =(String) tblResults.getModel().getValueAt(i,1);
        if (s.length()>0)
        {
            s +=",";
            s +=(String) tblResults.getModel().getValueAt(i,2) + ",";
            s +=String.valueOf( tblResults.getModel().getValueAt(i,3)) + ",";
            s +=String.valueOf( tblResults.getModel().getValueAt(i,4));
            writer.write(s);
            writer.newLine();
        }
     }

    writer.close();
    JOptionPane.showMessageDialog(marathon.this, "Data saved");
}
catch (IOException e)
```

Run the program. Enter a series of competitor names in the table, then click the "***Save file***" button.



Use ***Windows Explorer*** to locate the ***results.dat*** file in the ***marathon*** folder. Open this using a text editing application such as ***Notepad***. Records will only be stored for the rows of the table containing competitor names. Notice that commas mark the empty fields where times in hours and minutes have not yet been entered.



Close the program window and return to the NetBeans editing screen. Use the Design tab to move to the form layout screen, then double click the "***Load file***" button to create a method.

Add lines of code to call the ***clearTable( )*** method, then set up a ***TRY … CATCH*** structure which opens then closes the ***results.dat*** file.

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

    clearTable();
    try
    {
        FileReader r = new FileReader(filename);
        BufferedReader reader = new BufferedReader(r);

        reader.close();
    }
    catch (IOException e)
    {
         JOptionPane.showMessageDialog(marathon.this, "File error");
    }

}
```

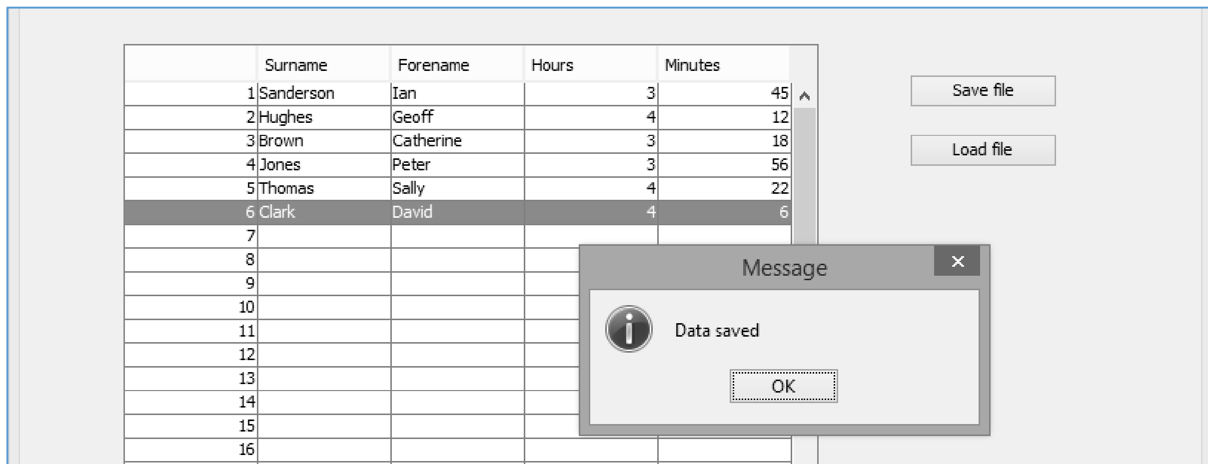
We can now insert lines of code to load each record in turn, split it into fields, then display the data in the table. The line

if (dataItem.length>2)

checks that hours and minutes fields have been saved for the current competitor, before attempting to display this data.

```
try

   {
       FileReader r = new FileReader(filename);
       BufferedReader reader = new BufferedReader(r);

       String s;
       int line=0;
       while((s=reader.readLine())!=null)
       {
               String dataItem[] = s.split(",");
               tblResults.getModel().setValueAt(dataItem[0],line,1);
               tblResults.getModel().setValueAt(dataItem[1],line,2);
               if (dataItem.length>2)
               {
                 tblResults.getModel().setValueAt(dataItem[2],line,3);
                 tblResults.getModel().setValueAt(dataItem[3],line,4);
               }
               line++;
       }

       reader.close();
   }
   catch (IOException e)
```

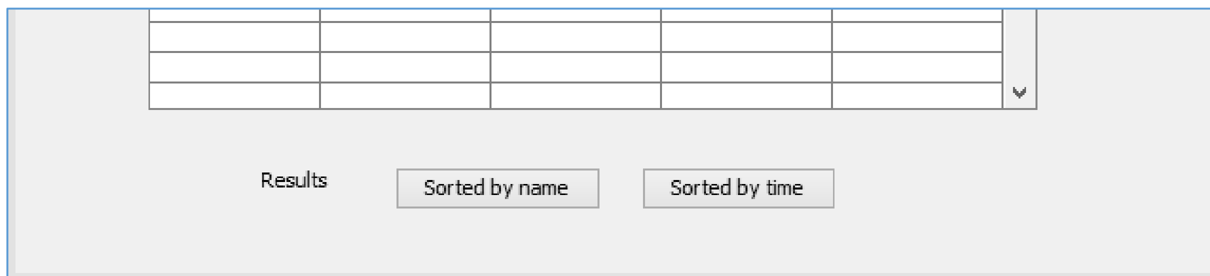
Run the program. click the "***Load file***" button and check that the competitor names are reloaded correctly.



Add times in hours and minutes for the competitors, then save the data.

Close, then re-run program. Click the "***Load file***" button and check that the competitor names and times are displayed correctly.

Close the program window and return to the NetBeans editing screen. Use the ***Design*** tab to move to the form layout view. Add a label "***Results***", and buttons with the captions "***Sorted by name***" and "***Sorted by time***". Give these buttons the names ***btnNameSort*** and ***btnTimeSort***.



Double click the "***Sorted by name***" button to create a method.

This method must begin by collecting the data values from the table and transferring them into arrays, ready for sorting. The same procedure will be needed when the "***Sorted by time***" button is clicked, so we can avoid duplicating the lines of program code by writing a separate method to collect the data from the table.

We will add a line of code to call a ***collectData( )*** method, and then on the lines below we will set up the ***collectData( )*** method:

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

     int count=collectData();

}

private int collectData()
{
     int count=0;

     return count;
}
```

You may notice that that ***collectData( )*** is being used in a different way to previous methods we have written, such as:

private void clearTable( )

The key word '***void***' means that the method will carry out some task then end, without the need to give back any result. In the case of ***clearTable( )***, all we require is that it resets the contents of the data table.

In some situations, however, we require a method to process data then give back a result value. For example, we might write a method to calculate the VAT on an item purchased in a shop. The first line of the method might be written:

private double VAT( )

The method will calculate the VAT amount, then return this as a number in ***double*** format.

It is important that the ***collectData( )*** method counts the number of competitors entered into the table. This value is returned as an integer number. Notice the structure of the method so far…

```
private int collectData( )
{
    int count=0;
    return count;
}
```

The value which will be returned is the variable ***count***. Initially this is set to zero, but it will be changed by further lines of code which we add to the method. The returned value is then transferred back to the line which originally called the method:

int count = collectData( );

and will be available for further use in the program. The returned value is called the ***output parameter*** of the method.

Add lines of code to the ***collectData( )*** method:

```
private int collectData()
{
    int count=0;

    TableCellEditor editor = tblResults.getCellEditor();
    if (editor != null)
    {
       editor.stopCellEditing();
    }
    for (int i=0;i<50;i++)
    {
       String h=tblResults.getModel().getValueAt(i,3).toString();
       if (h.length()>0)
       {
         surname[count]= tblResults.getModel().getValueAt(i,1).toString();
         forename[count]=tblResults.getModel().getValueAt(i,2).toString();
         hours[count]=Integer.parseInt(tblResults.getModel().getValueAt(i,3).toString());
         minutes[count]=Integer.parseInt(tblResults.getModel().getValueAt(i,4).toString());
         count++;
       }
     }

     return count;
```

We began by closing the table editor, to ensure that all data values were available for processing.

A loop checks each of the 50 rows of the table. Since the sorted lists will be the results for the marathon, we will only include competitors who completed the course and have a time recorded.

The program checks the entry in the ***Hours*** column. If this contains data, the values from each of the columns are collected and transferred into the arrays. ***Surname*** and ***Forename*** are stored in ***String*** format, and ***Hours*** and ***Minutes*** are in ***integer*** format. The ***count*** of competitors is then increased by one.

Return to the ***NameSort*** button click method. We will add the outer loop for the ***Bubble Sort***. The number of records to be sorted is given by the variable ***count***.

```
private void btnNameSortActionPerformed(java.awt.event.ActionEvent evt) {
     int count=collectData();

     Boolean swap=true;
     while (swap==true)
     {
         swap=false;
         for (int i=0; i<count-1;i++)
         {

         }
     }
  }
```

When sorting names alphabetically, both the surname and forenames should be used. People with the same surname should be sorted according to their forenames, for example:

Jones, Alun
Jones, Dafydd

We will therefore begin by combining surnames and forenames to make ***name*** variables for use in the sort procedure. A swap will be carried out if the names are not currently in the correct alphabetical order.

```
Boolean swap=true;
while (swap==true)
{
    swap=false;
    for (int i=0; i<count-1;i++)
    {
        String name1=surname[i]+" "+forename[i];
        String name2=surname[i+1]+" "+forename[i+1];
        if (name1.compareTo(name2)>0)
        {
           swap=true;
        }

    }
}
```

The final stage of the Bubble Sort is to carry out the ***triangular exchange*** of the array elements. In this program, it is necessary to make the same exchange of elements in each of the four arrays: ***surname[ ]***, ***forename[ ]***, ***hours[ ]*** and ***minutes[ ]***. If this is not done carefully, the data for different competitors will be mixed together and corrupted.

Add the lines of program to carry out the triangular exchange:

```
for (int i=0; i<count-1;i++)
{
    String name1=surname[i]+" "+forename[i];
    String name2=surname[i+1]+" "+forename[i+1];
    if (name1.compareTo(name2)>0)
    {
        swap=true;

        tempSurname=surname[i];
        surname[i]=surname[i+1];
        surname[i+1]=tempSurname;

        tempForename=forename[i];
        forename[i]=forename[i+1];
        forename[i+1]=tempForename;

        tempHours=hours[i];
        hours[i]=hours[i+1];
        hours[i+1]=tempHours;

        tempMinutes=minutes[i];
        minutes[i]=minutes[i+1];
        minutes[i+1]=tempMinutes;

    }
}
```

Once the sorting is complete, the data table can be cleared and the sorted records displayed. Add lines of code to do this.

```
                tempMinutes=minutes[i];
                minutes[i]=minutes[i+1];
                minutes[i+1]=tempMinutes;
            }
        }
    }
    clearTable();
    for(int i=0;i<count;i++)
    {

        tblResults.getModel().setValueAt(surname[i],i,1);
        tblResults.getModel().setValueAt(forename[i],i,2);
        tblResults.getModel().setValueAt(hours[i],i,3);
        tblResults.getModel().setValueAt(minutes[i],i,4);
    }

}
```

Run the program. Enter or reload a series of competitor names, then add race times. Leave some times blank, to represent competitors who did not complete the course. These competitors should not be included in the results list. You might also include more than one competitor with the same surname, to test the alphabetical sorting.

	Surname	Forename	Hours	Minutes
1	Sanderson	Ian	3	45
2	Hughes	Geoff	4	12
3	Brown	Catherine	3	18
4	Jones	Peter	3	56
5	Thomas	Sally	4	22
6	Clark	David	4	6
7	Smith	John		
8	Price	Steven		
9	Jones	Chris	3	42
10	Young	Mark	4	16
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Save file
Load file
Results
Sorted by name
Sorted by time

Click the "***Sorted by name***" button, and carefully check the results list which is displayed. The correct race time should be shown for each competitor, even though the order of the records has changed.

	Surname	Forename	Hours	Minutes
1	Brown	Catherine	3	18
2	Clark	David	4	6
3	Hughes	Geoff	4	12
4	Jones	Chris	3	42
5	Jones	Peter	3	56
6	Sanderson	Ian	3	45
7	Thomas	Sally	4	22
8	Young	Mark	4	16
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Save file
Load file
Results
Sorted by name
Sorted by time

Close the program and return to the NetBeans editing screen. Use the Design tab to go to the form layout view.

To complete the project, we will produce a method to sort the competitors' results according to their race times. We will only include competitors in the sorted list if they finished the marathon and their race time has been recorded.

Double click the "S***orted by time***" button to create a method. Begin the loops for the bubble sort in the same way as the ***Sorted by name*** method.

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

    int count=collectData();
    Boolean swap=true;
    while (swap==true)
    {
        swap=false;
        for (int i=0; i<count-1;i++)
        {

        }
    }

}
```

We will now calculate competitors' race times as ***total minutes***, then use these values for comparison in the Bubble Sort procedure:

```
while (swap==true)
{
    swap=false;
    for (int i=0; i<count-1;i++)
    {
        int time1=hours[i]*60+minutes[i];
        int time2=hours[i+1]*60+minutes[i+1];
        if (time1>time2)
        {
            swap=true;
            tempSurname=surname[i];
            surname[i]=surname[i+1];
            surname[i+1]=tempSurname;

            tempForename=forename[i];
            forename[i]=forename[i+1];
            forename[i+1]=tempForename;

            tempHours=hours[i];
            hours[i]=hours[i+1];
            hours[i+1]=tempHours;

            tempMinutes=minutes[i];
            minutes[i]=minutes[i+1];
            minutes[i+1]=tempMinutes;
        }

    }
}
```

Once the sorting is completed, the data table can be cleared and the sorted records displayed. Add lines of code to do this.

```
                    tempMinutes=minutes[i];
                    minutes[i]=minutes[i+1];
                    minutes[i+1]=tempMinutes;
                }
            }
        }

        clearTable();
        for(int i=0;i<count;i++)
        {

            tblResults.getModel().setValueAt(surname[i],i,1);
            tblResults.getModel().setValueAt(forename[i],i,2);
            tblResults.getModel().setValueAt(hours[i],i,3);
            tblResults.getModel().setValueAt(minutes[i],i,4);
        }

    }
```

Run the program. Load the data which you had previously saved on disc. Click the "***Sorted by time***" button and check that the competitors are listed correctly.

	Surname	Forename	Hours	Minutes
1	Brown	Catherine	3	18
2	Jones	Chris	3	30
3	Sanderson	Ian	3	45
4	Young	Karen	3	46
5	Jones	Peter	3	56
6	Clark	David	4	6
7	Hughes	Geoff	4	12
8	Thomas	Sally	4	22
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Save file

Load file

Results

Sorted by name

Sorted by time