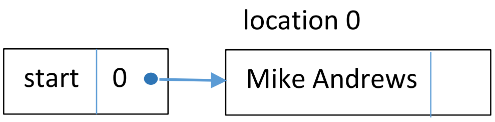
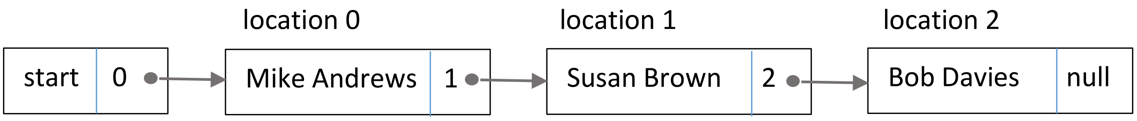
14 Linked lists

In a previous chapter we looked at two of the abstract data structures, ***stacks*** and ***queues***. We will now examine another abstract data structure, ***linked lists***.

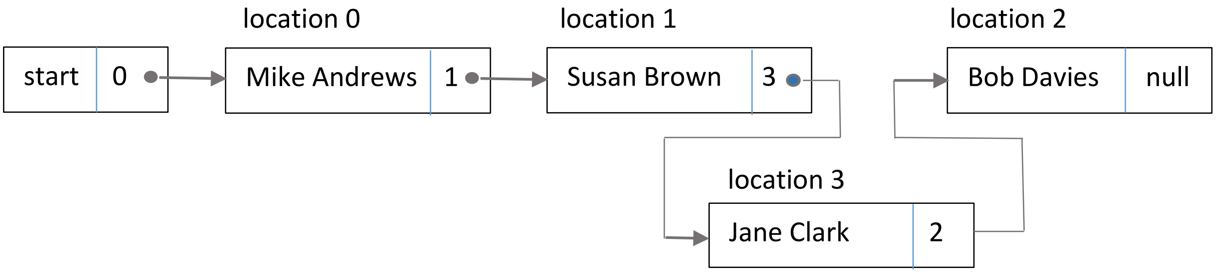
A linked list keeps a set of data in order by means of links between the data items. As an example, suppose that the secretary of a sports club wishes to keep records of members in order of ***surname***. A ***start pointer*** indicates the position of the first name:



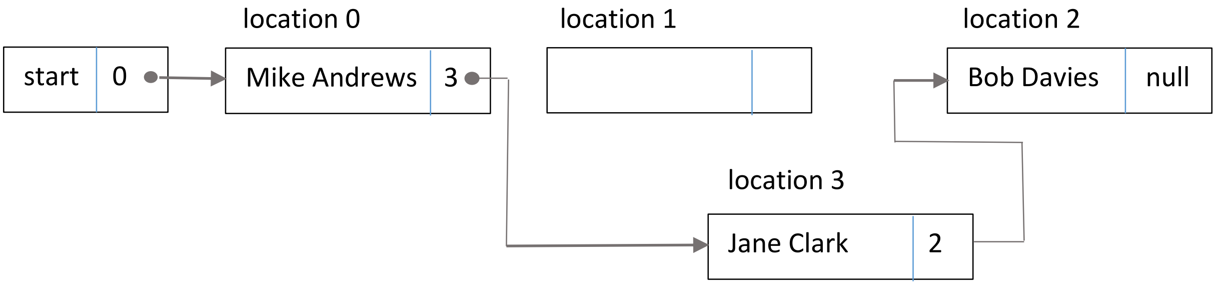
Further names can be linked by ***pointers in the records***. A ***null*** pointer value indicates the end of the sequence.



An important advantage of a linked list, compared to storing data in a simple array or file, is that a new record can be added at the correct point in the sequence without needing to move any other data. It is simply necessary to ***change the pointer value*** from the previous record in the sequence. For example, if Jane Clark joins the club:



Deleting records is equally straightforward. This only requires a single pointer value to be changed, with no need to move any of the data. The empty location can be re-used later. For example, if Susan Brown leaves the club:



We will develop a record keeping program for the sports club, demonstrating how records can be addded or deleted in the linked list.

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

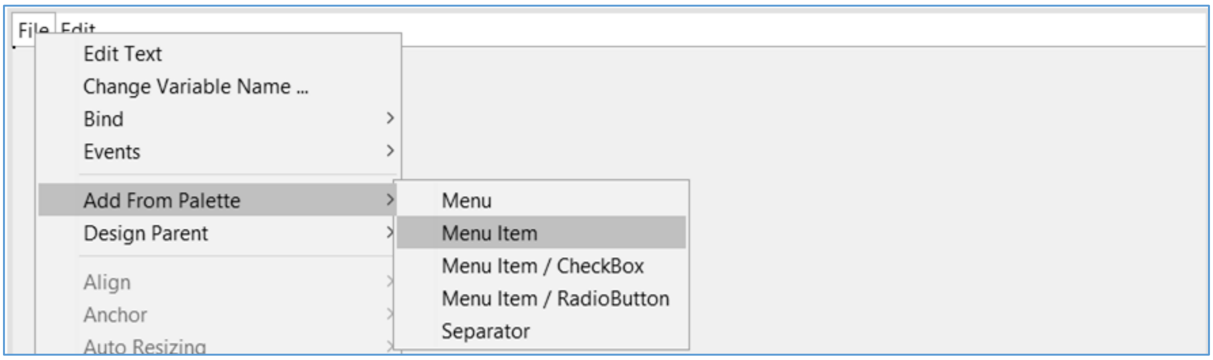
Return to the NetBeans editing page. Right-click on the ***linkedList*** project, and select ***New*** / ***JFrame Form***. Give the ***Class Name*** as ***linkedList***, and the ***Package*** as ***linkedListPackage***:

Return to the NetBeans editing screen.

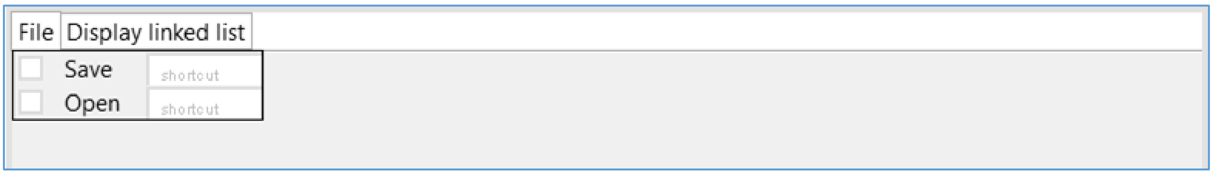
- Right-click on the ***form***, and select ***Set layout*** / ***Absolute layout***.
- Go to the ***Properties*** window on the bottom right of the screen and click the ***Code*** tab. Select the option: ***Form Size Policy*** / ***Generate pack()*** / ***Generate Resize code***.
- Click the Source 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.

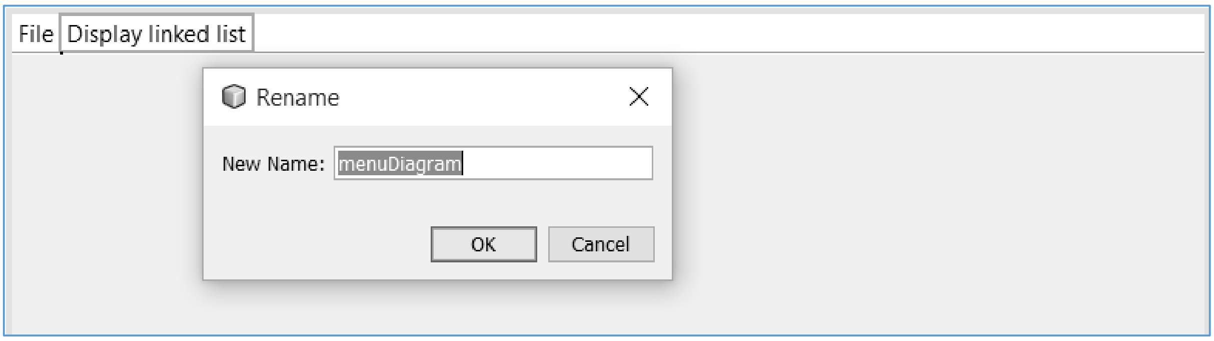
Select a ***Menu Bar*** component from the ***Palette*** and drag this onto the form. Right-click on the ***File*** menu option and select ***Add From Palette*** / ***Menu Item***. Right-click again on the ***File*** menu option and add a second ***Menu Item***.



Right-click on each of the menu items, and edit the text values to '***Save***' and '***Open***'. Change the text of the ***Edit*** menu option to '***Display linked list***'.

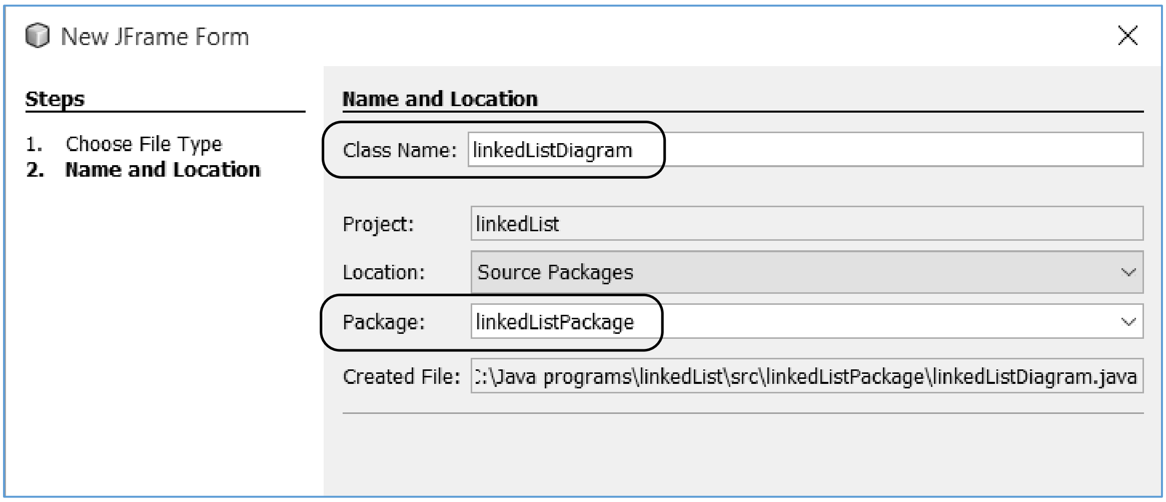


Right-click again on the menu options '***Save***', '***Open***' and '***Display linked list***'. Rename these as ***menuSave***, ***menuOpen*** and ***menuDiagram***.



New Name: menuDiagram

Create another form by right-clicking on ***linkedListPackage*** in the ***Projects*** window, then selecting ***New / JFrame Form***. Give the Class Name as '***linkedListDiagram***'. Leave the Package name as '***linkedListPackage***'.



New JFrame Form

Steps

1. Choose File Type
2. **Name and Location**

Name and Location

Class Name: linkedListDiagram

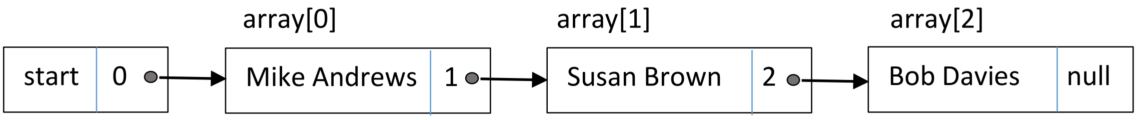
Project: linkedList

Location: Source Packages

Package: linkedListPackage

Created File: :\Java programs\linkedList\src\linkedListPackage\linkedListDiagram.java

A linked list can be represented by ***arrays***. Let us consider the example sequence:



To link these names in alphabetical order, it will be best to use separate parallel arrays for ***Surname*** and ***Forename***. An additional array will be needed to provide ***pointers*** to the next names in the sequence. We will use a pointer value of -1 to indicate that there are no further data items in the list.

	Surname	***Forename***	***Pointer***
[0]	Andrews	Mike	1
[1]	Brown	Susan	2
[2]	Davies	Bob	-1
[3]			

We will need to provide a ***start pointer*** with a value set to 0 to indicate the first data item in the sequence. Once we start to read the list, the pointer value associated with each name will then lead to the next name until the end of the list is reached.

Return to the ***linkedList.java*** form. Use the ***Source*** tab to open the program code page. Add definitions for the ***arrays*** and ***start pointer***, and also set a ***file name*** for saving the linked list on disc.

```
package linkedListPackage;

    public class linkedList extends javax.swing.JFrame {

        int startPointer=0;
        int[] pointer = new int[50];
        String[] surname=new String[50];
        String[] forename=new String[50];
        static String filename = "list.dat";

    public linkedList() {
```

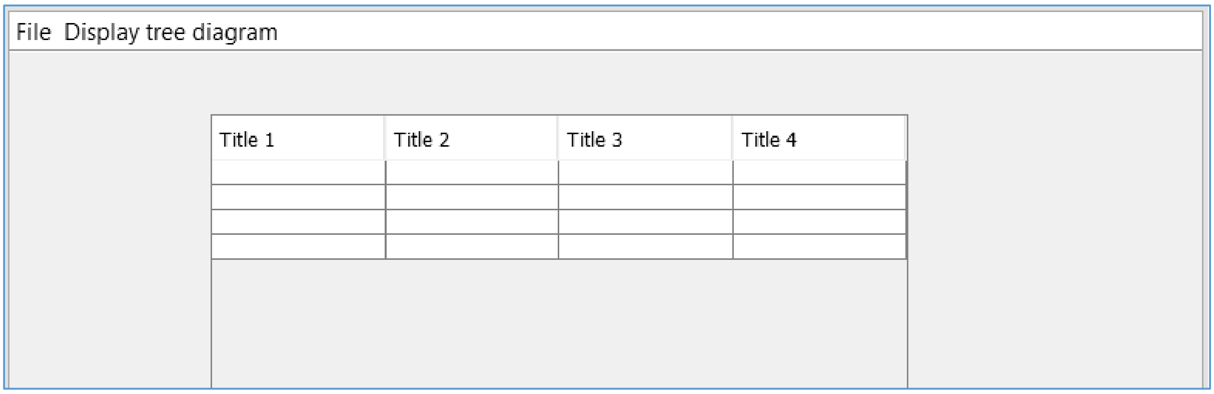
We will add the Java modules at the start of the program which will be needed for saving and reloading data files.

Complete the setting-up of the linked list arrays by using a loop to initialise all pointer values to -1.

```
package linkedListPackage;

import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;
import javax.swing.JOptionPane;


public class linkedList extends javax.swing.JFrame {

    int startPointer=0;
    int[] pointer = new int[50];
    String[] surname=new String[50];
    String[] forename=new String[50];
    static String filename = "list.dat";

    public linkedList() {
        initComponents();

        for (int i=0;i<50;i++)
        {
            pointer[i]=-1;
        }

    }
```

Click the ***Design*** tab to move to the form layout view.

We will be storing the linked list as a set of arrays, but the data can also be displayed on screen in a table. Add a ***Table*** component and rename this as ***tblLinkedList***.



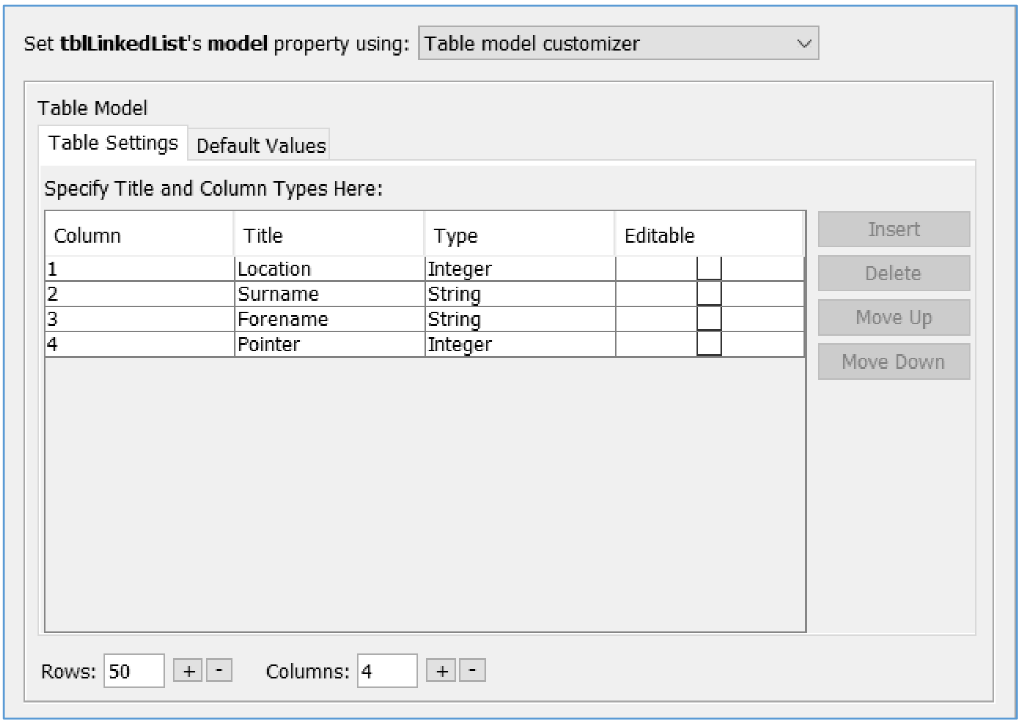
Title 1	Title 2	Title 3	Title 4

Select the Table and go to the ***Properties*** window. Locate the ***TableModel*** property and click in the right column to open the table editing window.

Set the number of ***Rows*** to **50** and ***Columns*** to **4**. Enter the column ***Titles*** and data ***Types***:

Location	***Integer***
Surname	***String***
Forename	***String***
Pointer	***Integer***

Remove the '***Editable***' tick from each of the columns.



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

Table Model

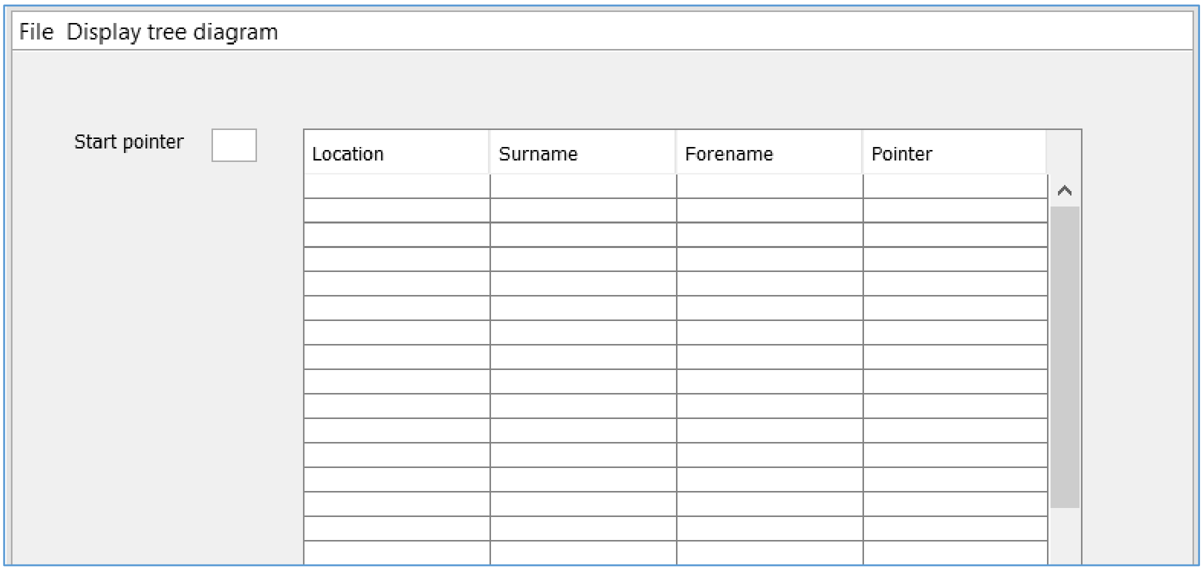
Specify Title and Column Types Here:

Column	Title	Type	Editable
1	Location	Integer	☐
2	Surname	String	☐
3	Forename	String	☐
4	Pointer	Integer	☐

Rows: 50 + - Columns: 4 + -

Click ***OK*** to return to the form display. Check that the column headings are shown correctly, and that the table includes a vertical scroll bar.

Add a label '***Start pointer***', with a ***text field*** alongside. Rename the text field as ***txtStartPointer***.



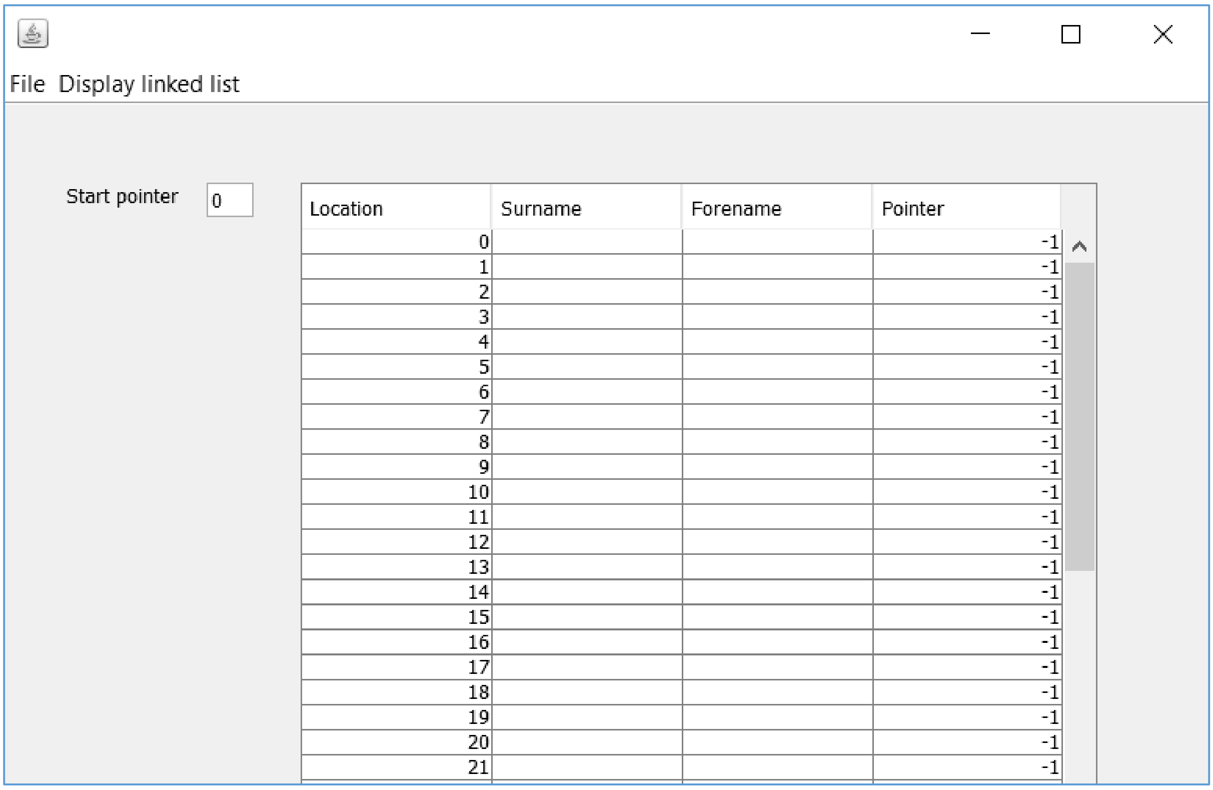
Start pointer

Location	Surname	Forename	Pointer

Use the ***Source*** tab to move back to the program code screen. Add a method below ***linkedList( )*** to display the ***surname***, ***forename*** and ***pointer*** array data in the table. We will also display the value of the ***start pointer***. Call the ***displayTable( )*** method from the ***linkedList( )*** method.

```
public linkedList() {
    initComponents();
    for (int i=0;i<50;i++)
    {
        pointer[i]=-1;
    }

    displayTable();

}

private void displayTable()
{
    for (int i=0;i<50;i++)
    {
        tblLinkedList.getModel().setValueAt(i,i,0);
        tblLinkedList.getModel().setValueAt(surname[i],i,1);
        tblLinkedList.getModel().setValueAt(forename[i],i,2);
        tblLinkedList.getModel().setValueAt(pointer[i],i,3);

    }
    txtStartPointer.setText(String.valueOf(startPointer));
}
```

Run the program. No names have been entered yet, so the Surname and Forename columns will be blank. All pointers were initialised to -1.



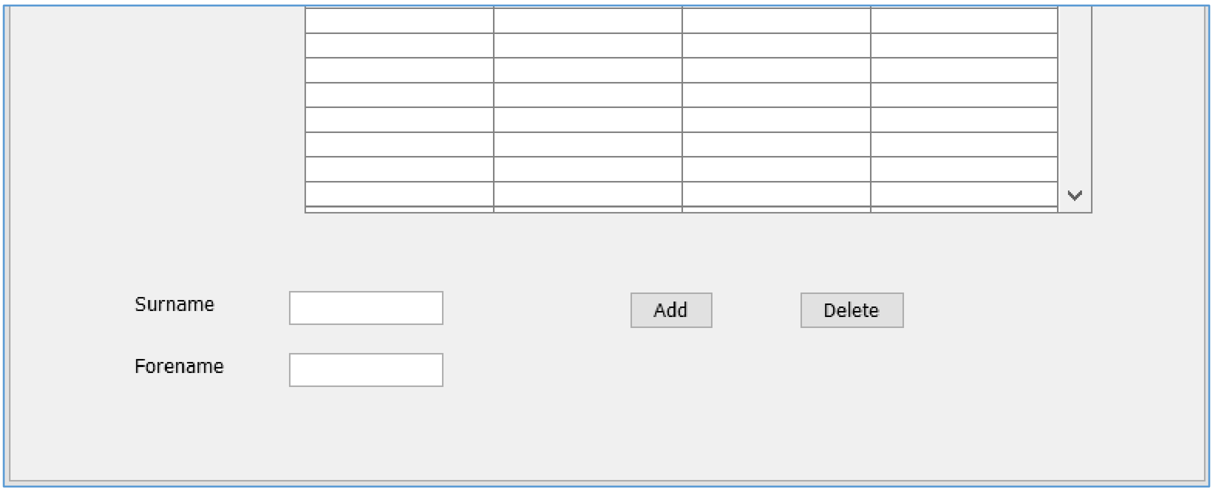
Start pointer 0

Location	Surname	Forename	Pointer
0			-1
1			-1
2			-1
3			-1
4			-1
5			-1
6			-1
7			-1
8			-1
9			-1
10			-1
11			-1
12			-1
13			-1
14			-1
15			-1
16			-1
17			-1
18			-1
19			-1
20			-1
21			-1

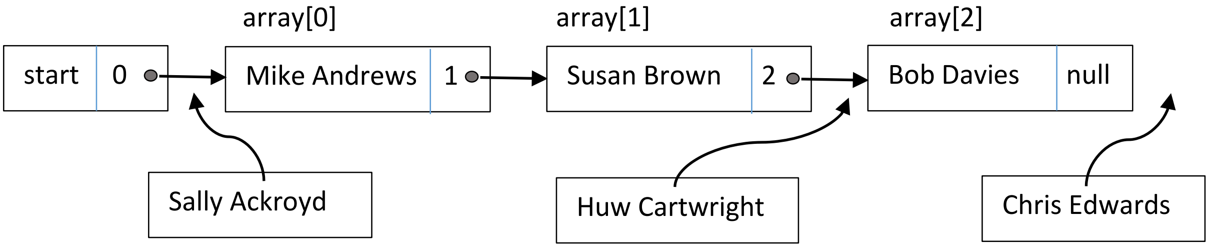
Close the program and return to the NetBeans editing screen. Use the Design tab to move to the form layout view.

Add components below the Table:

- A label '***Surname***' with a text field alongside. Rename the text field as ***txtSurname***.
- A label '***Forename***' with a text field alongside. Rename the text field as ***txtForename***.
- Buttons with the captions '***Add***' and '***Delete***'. Rename the buttons as ***btnAdd*** and ***btnDelete***.



We can now begin the design of an algorithm for adding names to the linked list. Let us think about this task in more detail, using the example list:



Three different situations could occur:

- The new record could come ***before the first record*** in the current sequence, as in the case of ***Sally Ackroyd***.
- The new record may need to be inserted ***between two records*** in the current sequence, as with ***Huw Cartwright***.
- The new record could come ***after the last record*** in the sequence, as with ***Chris Edwards***.

We will need to program each of these cases separately.

Begin by double-clicking the '***Add***' button to create a method. We will first add lines of code, as shown below, to collect the surname and forename from the text fields. The program will carry out a presence check to ensure that text values have been entered, then call the ***add( )*** method which we will insert immediately after the button click method.

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

        String surnameEntered=txtSurname.getText();
        String forenameEntered=txtForename.getText();
        if(surnameEntered.length()>0 && forenameEntered.length()>0)
        {
           add(surnameEntered,forenameEntered);
        }
        txtSurname.setText("");
        txtForename.setText("");

    }

    private void add(String surnameEntered, String forenameEntered)
    {

    }
```

We will set up two variables, ***previous*** and ***current***, which will be useful when we search the linked list to find the correct position to insert a new record. Both of these will be initialised to the value of the ***start pointer***.

We will first check for the case where the linked list is empty. This will be true if the start pointer has set the value of ***current*** to point to an empty surname. Add lines of code to detect this situation.

```
    private void add(String surnameEntered, String forenameEntered)
    {

       int previous=startPointer;
       int current=startPointer;
       Boolean finished;
       String nameEntered = surnameEntered + " " + forenameEntered;
       String name;

       if (surname[current]==null)
       {
          emptyList(current);
       }
       displayTable();

    }
```

We will now design an ***emptyList( )*** method to handle the data and pointer values.

For an ***empty list***, we simply store the surname and forename in the array position indicated by the start pointer, and leave the pointer value of the new record as -1 to mark the end of the list. For example, if we add ***Hughes, Steven*** to an empty list, the data should be stored as:

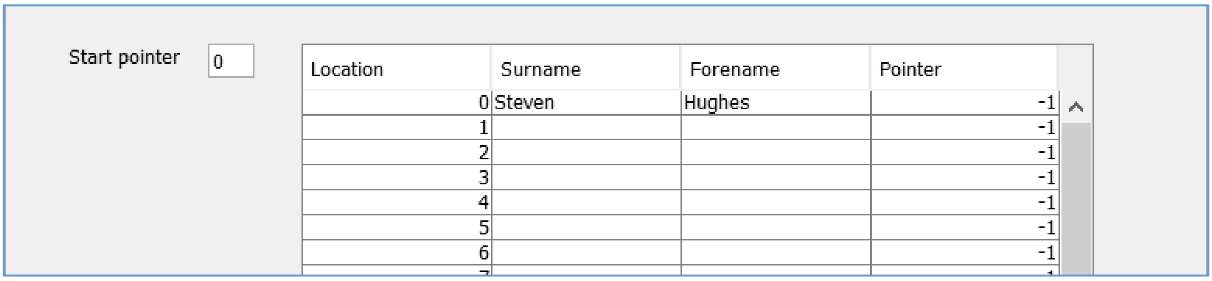
Start pointer 0

	Surname	***Forename***	***Pointer***
[0]	Hughes	Steven	-1
[1]			
[2]			
[3]			

Insert the ***emptyList( )*** method immediately after the ***add( )*** method.

```
        if (surname[current]==null)
        {
           emptyList(current);
        }
        displayTable();
    }

 private void emptyList(int current)
 {
    surname[current]=txtSurname.getText();
    forename[current]=txtForename.getText();
 }
```

Run the program. Enter '***Hughes***' into the ***Surname*** text field and '***Steven***' into the ***Forename*** text field, then click the ***Add*** button. Check that the name appears correctly in the table at ***Location 0***, and that the ***Pointer*** value is still **-1**.



Start pointer 0

Location	Surname	Forename	Pointer
0	Steven	Hughes	-1
1			-1
2			-1
3			-1
4			-1
5			-1
6			-1

Close the program window and return to the NetBeans editing screen. The next situation to consider is that the list already contains data, but the new name to be added comes before the current first name in the sequence. For example, if the list already contains Stephen Hughes:

Start pointer 0

	Surname	***Forename***	***Pointer***
[0]	Hughes	Steven	-1
[1]			
[2]			
[3]			

and we now add ***Peter Edwards***, the situation should become:

Start pointer 1

	Surname	***Forename***	***Pointer***
[0]	Hughes	Steven	-1
[1]	Edwards	Peter	0
[2]			
[3]			

Notice that the record for ***Edwards, Peter*** has been stored in the next available empty location, and the ***start pointer*** has been changed to point to this location. The pointer from ***Peter Edwards*** now links to the record in position 0.

Add lines of code to the ***add( )*** method to detect the situation where a new record needs to be inserted before the current first record of the linked list. We use the ***compareTo( )*** function to check whether the name at the start of the list comes later in the alphabet than the name which has just been entered.

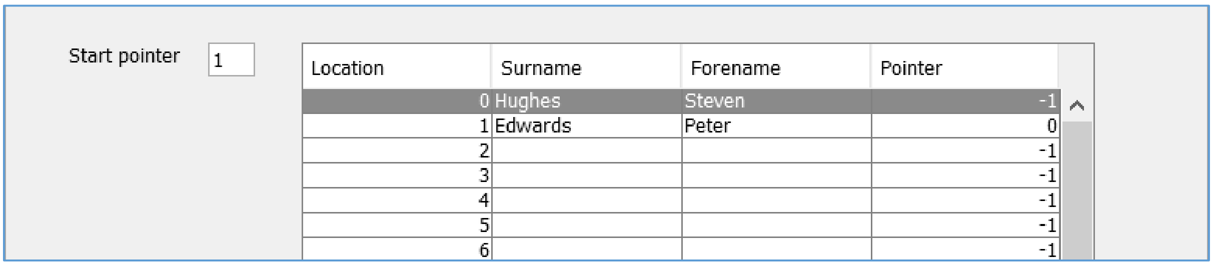
```
private void add(String surnameEntered, String forenameEntered)
{
    int previous=startPointer;
    Boolean finished;
    String nameEntered=surnameEntered+" "+forenameEntered;
    String name;
    int current=startPointer;

   if (surname[current]==null)
   {
       emptyList(current);
   }
   else
   {
       name=surname[current]+" "+forename[current];
       if (name.compareTo(nameEntered)>0)
       {
           startList(current);
       }
   }
}
```

Create a ***startList( )*** method immediately below the ***add( )*** method to handle the changes to the data and pointers.

We will also require a function ***findEmpty( )***, to locate the first empty group of array elements where our new name can be stored. When ***findEmpty( )*** is called, it will check the ***surname array*** and give back the number of the first empty location that it finds.

```
private void startList(int current)
{
    int empty=findEmpty();
    surname[empty]=txtSurname.getText();
    forename[empty]=txtForename.getText();
    pointer[empty]=current;
    startPointer=empty;
}

private int findEmpty()
{
    int location=0;
    while(surname[location]!=null)
    {
       location++;
    }
    return location;
}
```

Run the program. Enter the name ***Steven Hughes***, followed by the name ***Peter Edwards***. Check that the start pointer and the pointer from Peter Edwards have been set correctly.



Start pointer 1

Location	Surname	Forename	Pointer
0	Hughes	Steven	-1
1	Edwards	Peter	0
2			-1
3			-1
4			-1
5			-1
6			-1

Close the program window and return to the NetBeans editing screen. We will now consider the case where the new name to be added comes after all the existing names, and should be added to the end of the linked list. For example, suppose that the list already contains two names:

Start pointer 1

	Surname	***Forename***	***Pointer***
[0]	Hughes	Steven	-1
[1]	Edwards	Peter	0
[2]			
[3]			

We will now add another name, ***Susan Morris***. The situation should now become:

Start pointer 1

	Surname	***Forename***	***Pointer***
[0]	Hughes	Steven	2
[1]	Edwards	Peter	0
[2]	Morris	Susan	-1
[3]			

The new name has been stored in the first available space, at ***location 2***. The pointer from the previous last item in the list, ***Steven Hughes***, now points to ***Susan Morris***. The pointer from Susan Morris still has a value of **-1**, as this name is the final item in the updated list.

We will now insert lines of code into the ***add( )*** method, as shown below, to handle the situation where a new record becomes the final item of the linked list.

A ***WHILE… loop*** checks each of the existing records, and uses the pointer value to update the ***current*** variable and move on to the next record. When a pointer value of -1 is reached, the end of the list is detected and the loop ends. We then call a method ***endList( )*** to add the new name to the existing list at this point.

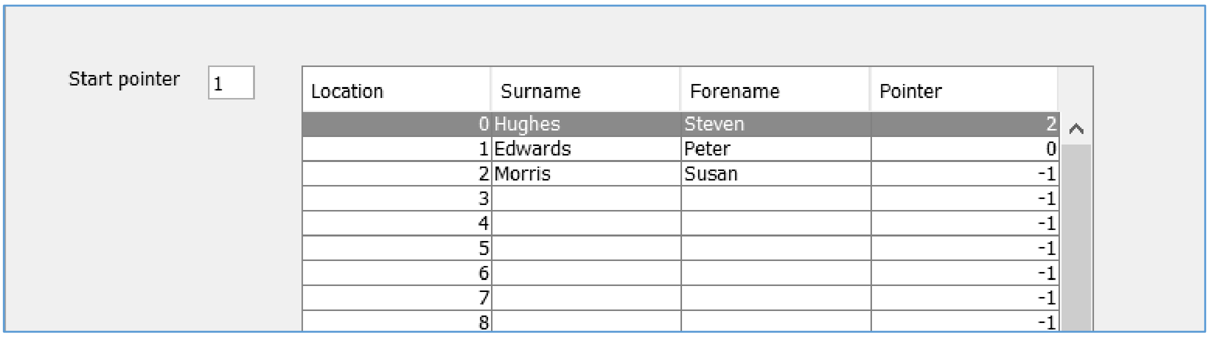
Create the ***endList( )*** method after the ***add( )*** method, to handle the changes to the data and pointers.

```
        if (surname[current]==null)
        {
            emptyList(current);
        }
        else
        {
            name=surname[current]+" "+forename[current];
            if (name.compareTo(nameEntered)>0)
            {
                startList(current);
            }
            else
            {
                finished=false;
                while(finished==false)
                {
                    if(pointer[current]<0)
                    {
                        finished=true;
                        endList(current);
                    }
                    else
                    {
                       current=pointer[current];
                    }
                }
            }
         }
         displayTable();
    }

    private void endList(int current)
    {
         int empty=findEmpty();
         surname[empty]=txtSurname.getText();
         forename[empty]=txtForename.getText();
         pointer[current]=empty;
    }
```

Run the program. Enter the names: ***Steven Hughes***, ***Peter Edwards*** and ***Susan Morris***, in that order.

- ***Steven Hughes*** should start the empty list.
- ***Peter Edwards*** becomes the new first list item, then
- ***Susan Morris*** is added to the end of the list.

Check that the pointers are updated correctly.



Start pointer 1

Location	Surname	Forename	Pointer
0	Hughes	Steven	2
1	Edwards	Peter	0
2	Morris	Susan	-1
3			-1
4			-1
5			-1
6			-1
7			-1
8			-1

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

We have one last situation to consider, when the new name needs to be inserted ***within*** the existing linked list sequence. Suppose that the list currently contains three names:

Start pointer 1

	Surname	***Forename***	***Pointer***
[0]	Hughes	Steven	2
[1]	Edwards	Peter	0
[2]	Morris	Susan	-1
[3]			

We will now add ***Sarah Green***. The new name needs to be inserted between ***Edwards*** and ***Hughes*** in the sequence. The surname and forename for ***Sarah Green*** can be added at the next available location 3.

Start pointer 1

	Surname	***Forename***	***Pointer***
[0]	Hughes	Steven	2
[1]	Edwards	Peter	3
[2]	Morris	Susan	-1
[3]	Green	Sarah	0

Notice that the pointers have been updated. ***Peter Edwards*** now links to ***Sarah Green*** at ***location 3***, and ***Sarah Green*** links to ***Steven Hughes*** at ***location 0***.

Complete the ***add( )*** method by inserting lines of code to detect the situation where a name should be inserted ***within*** the list. The program checks each existing name in the linked list and stops if it finds a name later in the alphabet.

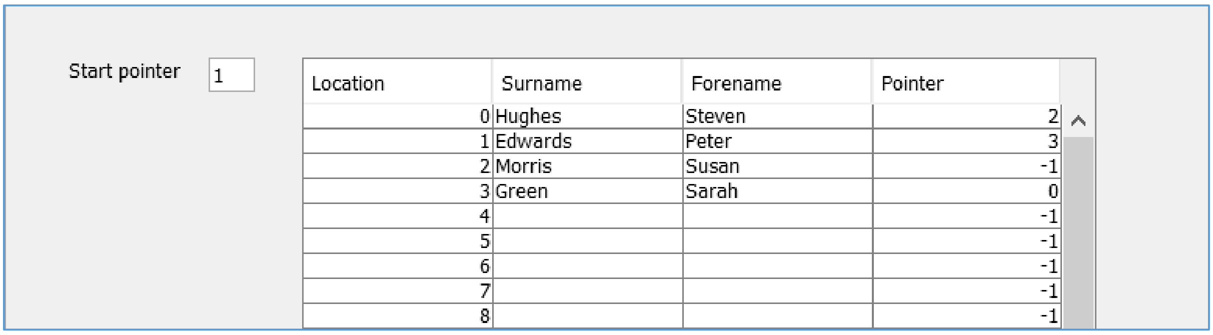
Notice that we use the ***previous*** variable to keep track of the previous record location examined, so that pointers can be set correctly when the new name is inserted.

```
           while(finished==false)
           {
               if(pointer[current]<0)
               {
                   finished=true;
                   endList(current);
               }
               else
               {
                  previous=current;

                  current=pointer[current];

                  name=surname[current]+" "+forename[current];
                  if (name.compareTo(nameEntered)>0)
                  {
                      finished=true;
                      midList(current,previous);
                  }

               }
           }
       }
   }
   displayTable();
}
```

Create the ***midList( )*** method after the ***add( )*** method, to handle the changes to the data and pointers.

```
private void midList(int current, int previous)
{
    int empty=findEmpty();
    surname[empty]=txtSurname.getText();
    forename[empty]=txtForename.getText();
    pointer[previous]=empty;
    pointer[empty]=current;
}
```

Run the program. Enter the names: ***Steven Hughes***, ***Peter Edwards***, ***Susan Morris*** and ***Sarah Green*** in that order. Check that the pointers to and from ***Sarah Green*** have been set correctly.



Start pointer 1

Location	Surname	Forename	Pointer
0	Hughes	Steven	2
1	Edwards	Peter	3
2	Morris	Susan	-1
3	Green	Sarah	0
4			-1
5			-1
6			-1
7			-1
8			-1

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

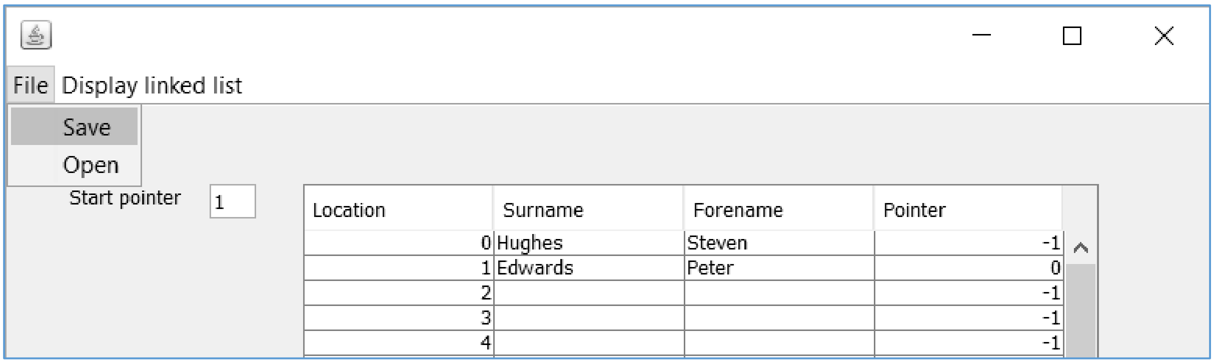
We have completed the basic functions for adding records at any point in the linked list, so this would be a good time to set up methods to save the linked list data on disc, and reload the saved records.

Use the ***Design*** tab to change to the form layout view. Go to the ***Menu Bar*** at the top of the form and double-click the '***Save***' menu option to create a method.

Add code to delete any previous data file, then open a new file. We will begin by saving the start pointer value in text format as a two character string.

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

    File oldfile = new File(filename);
    oldfile.delete();
    try (RandomAccessFile file = new RandomAccessFile(filename, "rw"))
    {
        String tStart=String.format("%-2s", startPointer);
        String s = tStart + "***";
        file.write(s.getBytes());
        file.close();
    }
    catch(IOException e)
    {
        JOptionPane.showMessageDialog(linkedList.this, "File error");
    }
}
```

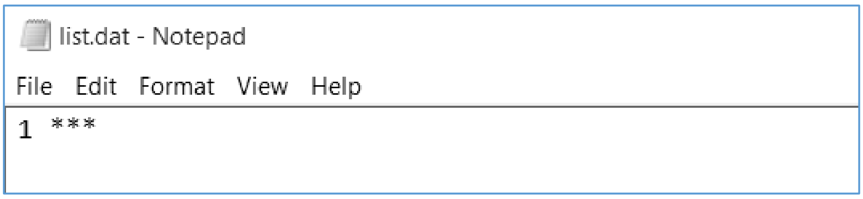
Run the program. Enter several names, then click the ***Save*** menu option.



Start pointer 1

Location	Surname	Forename	Pointer
0	Hughes	Steven	-1
1	Edwards	Peter	0
2			-1
3			-1
4			-1

Use Windows Explorer to locate the ***list.dat*** file in the ***linkedList*** project folder. Open the file with a text editing application such as Notepad, and check that the file contains the correct ***start pointer*** value.



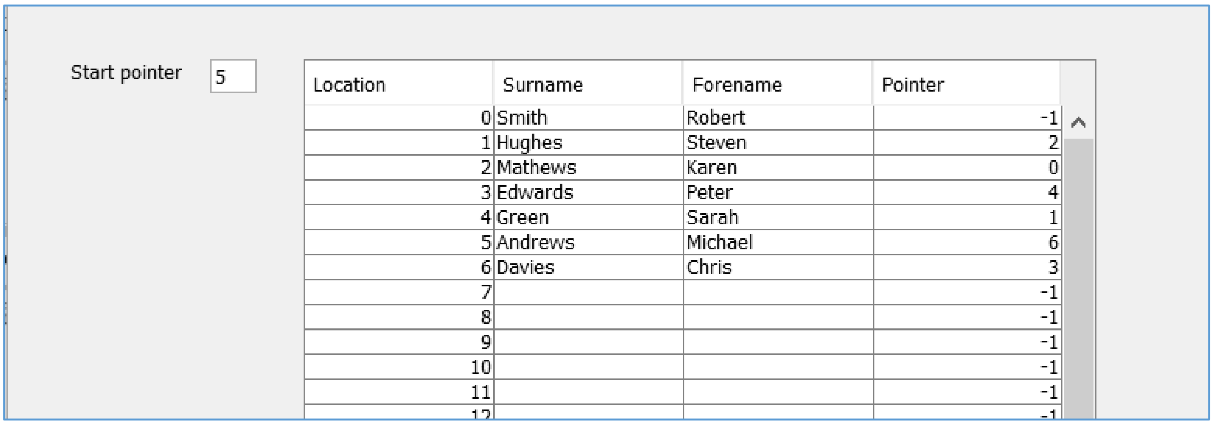
```
1 ***
```

Close the program window and return to the source code editing screen.

We will now use a ***loop*** to save each of the data entries. ***Fixed length records*** can be used, with the ***location numbers*** and ***pointers*** stored in text form as 2 characters, and the ***surnames*** and ***forenames*** stored as 24 character strings.

```
try (RandomAccessFile file = new RandomAccessFile(filename, "rw"))
{
    String tStart=String.format("%-2s", startPointer);
    String s = tStart + "***";
    file.write(s.getBytes());

    for (int i=0; i<50; i++)
    {
        if (surname[i]!=null)
        {
            String tLocation=String.format("%-2s", i);
            String tSurname=String.format("%-24s", surname[i]);
            String tForename=String.format("%-24s", forename[i]);
            String tPointer=String.format("%-2s", pointer[i]);
            s = tLocation + tSurname + tForename + tPointer + "***";
            file.write(s.getBytes());
        }
    }

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

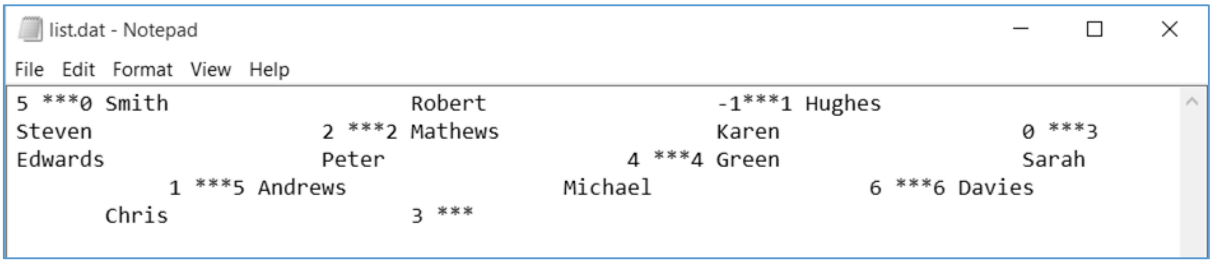
Run the program. Add a series of names in any random order, then check that the pointer values produce a correct linked list in alphabetical order.



Start pointer 5

Location	Surname	Forename	Pointer
0	Smith	Robert	-1
1	Hughes	Steven	2
2	Mathews	Karen	0
3	Edwards	Peter	4
4	Green	Sarah	1
5	Andrews	Michael	6
6	Davies	Chris	3
7			-1
8			-1
9			-1
10			-1
11			-1
12			-1

Click the '***Save***' menu option. Use a text editing application such as Notepad to open the ***list.dat*** file in the ***linkedList*** project folder. Check that the data in the file corresponds to the names and pointers shown in the table.



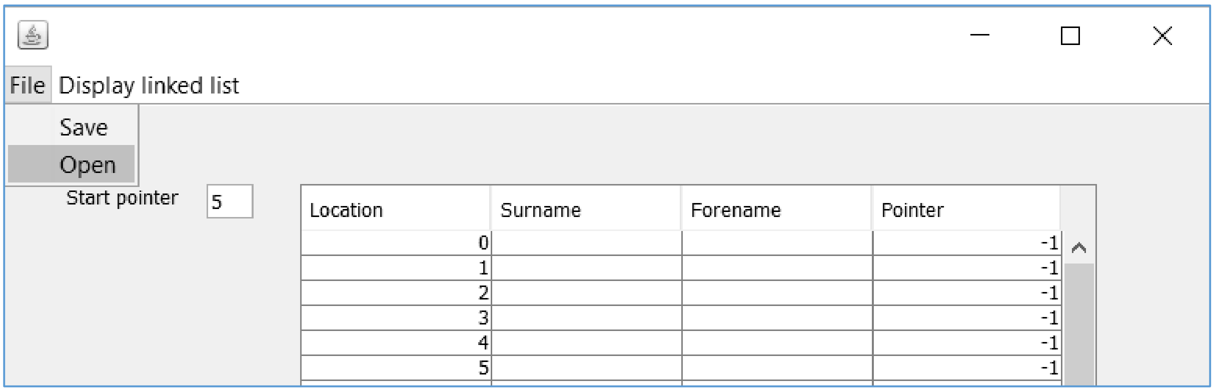
```
5 ***0 Smith                                 Robert                      -1***1 Hughes
Steven                               2 ***2 Mathews                      Karen                        0 ***3
Edwards                              Peter                        4 ***4 Green                        Sarah
                1 ***5 Andrews                            Michael                      6 ***6 Davies
       Chris                                 3 ***
```

Close the program window and return to the NetBeans editing screen. We will now produce a method to reload the linked list data.

Use the ***Design*** tab to select the form layout view, then double-click the '***Open***' option from the ***Menu Bar*** to create a method. Begin by adding code which will load the ***startPointer***.

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

    String tStart;
    String s;
    try
    {
        int position;
        RandomAccessFile file = new RandomAccessFile(filename, "r");
        byte[] bytes = new byte[2];
        file.read(bytes);
        s=new String(bytes);
        tStart=s.substring(0,2).trim();
        startPointer=Integer.valueOf(tStart);
        file.close();
        displayTable();
    }
    catch(IOException e)
    {
        JOptionPane.showMessageDialog(linkedList.this, "File error");
    }

}
```

Run the program. Go to the menu bar and click the '***Open***' option. Check that the start pointer value saved earlier is displayed correctly.



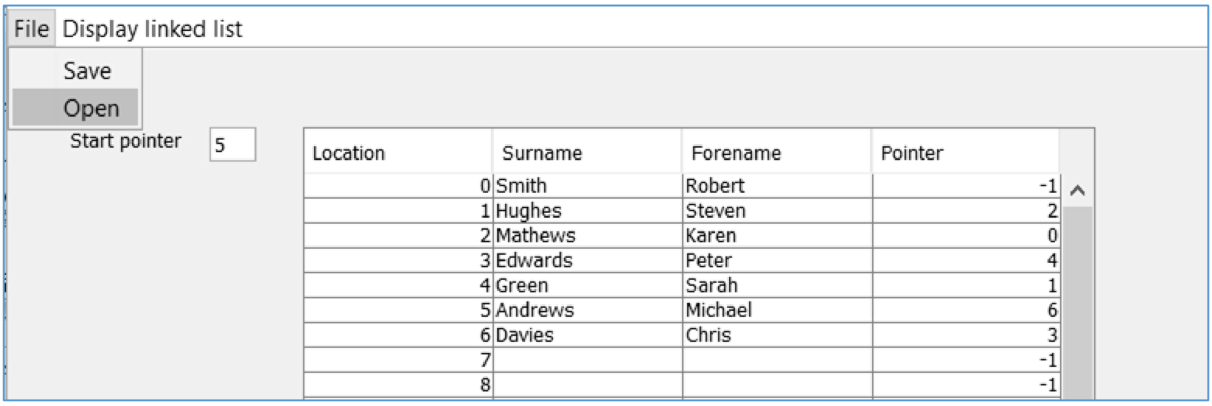
Start pointer 5

Location	Surname	Forename	Pointer
0			-1
1			-1
2			-1
3			-1
4			-1
5			-1

Close the program window and return to the code editing screen. Add a block of code to the '***Open***' method which will load each of the records, split the data into the ***location***, ***surname***, ***forename*** and ***pointer*** fields, then copy these back into the set of arrays. Each fixed length record has a total length of ***55 bytes***, and this value is used to calculate the number of records in the file.

```
try
{
    int position;

    String tLocation;
    String tSurname;
    String tForename;
    String tPointer;

    RandomAccessFile file = new RandomAccessFile(filename, "r");
    byte[] bytes = new byte[2];
    file.read(bytes);
    s=new String(bytes);
    tStart=s.substring(0,2).trim();
    startPointer=Integer.valueOf(tStart);

    int readingCount=(int) (file.length()-5) /55;
    for (int i=0; i<readingCount; i++)
    {
        position=i*55 + 5;
        file.seek(position);
        bytes = new byte[55];
        file.read(bytes);
        s=new String(bytes);
        tLocation=s.substring(0,2).trim(); s=s.substring(2);
        tSurname=s.substring(0,24).trim(); s=s.substring(24);
        tForename=s.substring(0,24).trim(); s=s.substring(24);
        tPointer=s.substring(0,2).trim();
        int n=Integer.valueOf(tLocation);
        surname[n]=tSurname;
        forename[n]=tForename;
        pointer[n]=Integer.valueOf(tPointer);
    }

    file.close();
    displayTable();
}
catch(IOException e)
```

Run the program. Click the '***Open***' option and check that the full sequence of records are displayed correctly in the table.



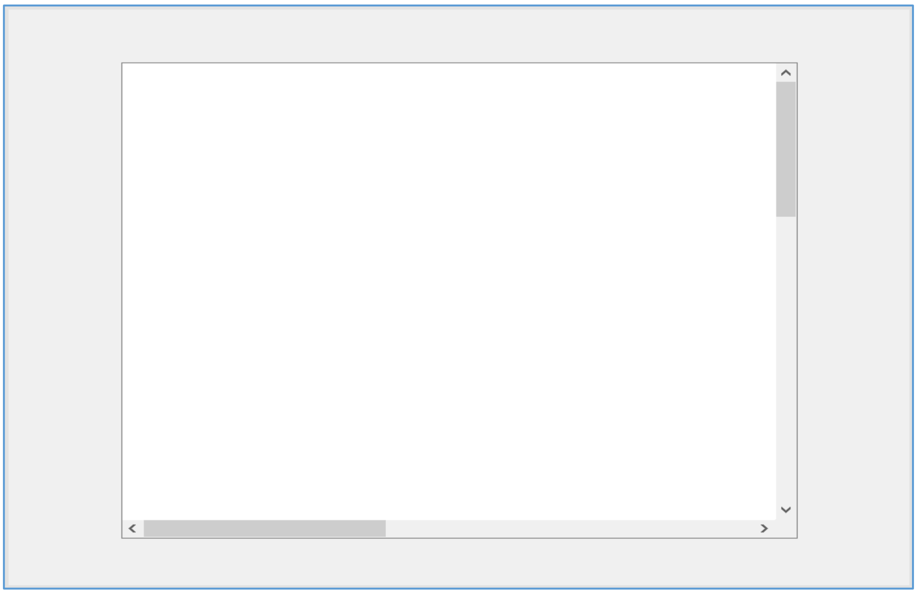
Start pointer 5

Location	Surname	Forename	Pointer
0	Smith	Robert	-1
1	Hughes	Steven	2
2	Mathews	Karen	0
3	Edwards	Peter	4
4	Green	Sarah	1
5	Andrews	Michael	6
6	Davies	Chris	3
7			-1
8			-1

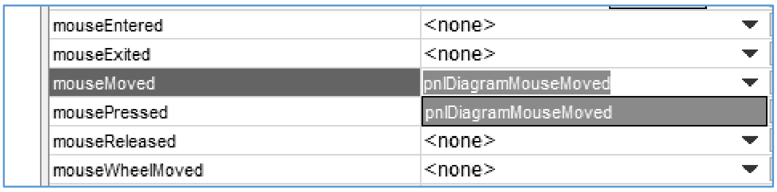
Close the program window and return to the NetBeans editing screen.

We are able to create, save and reload the linked list, but it is not very easy to examine the data in alphabetical order. In the next section, ***we will use graphics to display the linked list with the names shown in their correct order.***

Use the tab above the editing window to move to the ***linkedListDiagram.java*** page. Click the ***Design*** tab to move to the form layout view. Add a ***Scroll Pane*** component to the form, and drag this to a suitable size. Select a ***Panel*** component, then drag and drop this in the middle of the scroll pane. Rename the panel as ***pnlDiagram***. Set the ***background*** property to ***White***, and the ***preferredSize*** property to ***[1600,1600]***. Vertical and horizontal scroll bars should appear on the panel. Right-click on the panel and select ***Layout*** / ***Absolute Layout***.



Click the ***Events*** tab at the top of the Properties window, then locate the ***mouseMoved*** event. Select ***pnlDiagramMouseMoved*** from the drop down list.



mouseEntered	<none>
mouseExited	<none>
mouseMoved	pnlDiagramMouseMoved
mousePressed	pnlDiagramMouseMoved
mouseReleased	<none>
mouseWheelMoved	<none>

Add a line of code to the ***pnlDiagramMouseMoved*** method to call a ***drawDiagram( )*** method.

```
private void pnlDiagramMouseMoved(java.awt.event.MouseEvent evt) {
   drawDiagram();
}
```

Scroll to the top of the program listing and add Java modules which will be needed to create graphics. We will also add arrays to hold the linked list data, and begin the ***drawDiagram( )*** method.

```
package linkedListPackage;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;

public class linkedListDiagram extends javax.swing.JFrame {

    int startPointer;
    int[] pointer = new int[50];
    String[] surname=new String[50];
    String[] forename=new String[50];

    public linkedListDiagram() {
        initComponents();
    }

    private void drawDiagram()
    {

    }
```

At this point we have a slight problem! In order to display the linked list, the ***drawDiagram( )*** method will need access to all the linked list data. Unfortunately, this data is held in the table and arrays on the previous form and is not yet available to the ***linkedListDiagram.java*** page.

A simple way to transfer data from one form to another is to use a ***data*** class as a temporary storage area. Go to the ***Projects*** window at the top left of the screen and locate the ***linkedListPackage*** folder. Right-click on ***linkedListPackage*** and select ***New*** / ***Java Class***. Give the Class Name as '***data***', leaving the Package name as '***linkedListPackage***'.

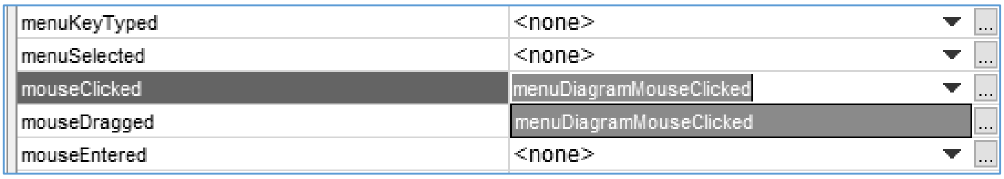
Add variables to the ***data*** class to hold the linked list data items and start pointer value.

```
package linkedListPackage;

    public class data {

        public static int startPointer;
        public static String[] surname=new String[50];
        public static String[] forename=new String[50];
        public static int[] pointer = new int[50];

    }
```

We will need to save the linked list into the ***data*** class when the '***Display linked list***' option is selected from the Menu Bar of the main program.

Use the tab at the top of the editing window to move to the ***linkedList.java*** page.

Select the '***Display linked list***' menu item, go to the Properties window and click the ***Events*** tab. Locate the ***mouseClicked*** event, then accept ***menuDiagramMouseClicked*** from the drop down list.



menuKeyTyped	<none>
menuSelected	<none>
mouseClicked	menuDiagramMouseClicked
mouseDragged	menuDiagramMouseClicked
mouseEntered	<none>

The ***menuDiagramMouseClicked( )*** method will open.

We will add a line of code to call a ***saveData( )*** method, then add this below ***menuDiagramMouseClicked( )***.

```
private void menuDiagramMouseClicked(java.awt.event.MouseEvent evt) {

    saveData();
    new linkedListDiagram().setVisible(true);

}

private void saveData()
{
    data.startPointer=startPointer;
    for (int i=0;i<50;i++)
    {
        data.surname[i]=surname[i];
        data.forename[i]=forename[i];
        data.pointer[i]=pointer[i];
    }
}
```

Return to the ***linkedListDiagram.java*** page. Locate the ***drawDiagram( )*** method and add a line of code to call a ***loadData( )*** method.

```
private void drawDiagram()
{

    loadData();

}
```

Add the ***loadData( )*** method below the ***drawDiagram( )*** method.

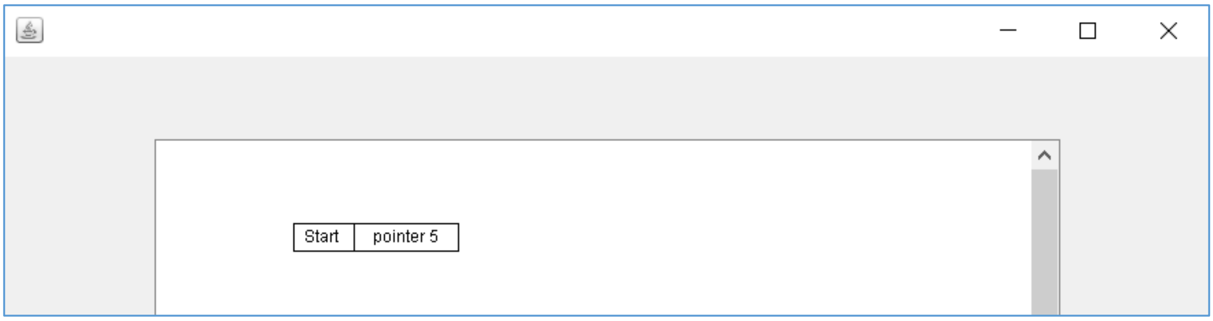
```
    private void drawDiagram()
    {
        loadData();
    }

    private void loadData()
    {
        startPointer=data.startPointer;
        for (int i=0;i<50;i++)
        {
            surname[i]=data.surname[i];
            forename[i]=data.forename[i];
            pointer[i]=data.pointer[i];
        }
    }
```

We now have access to the necessary data, and we can continue with drawing the linked list diagram.

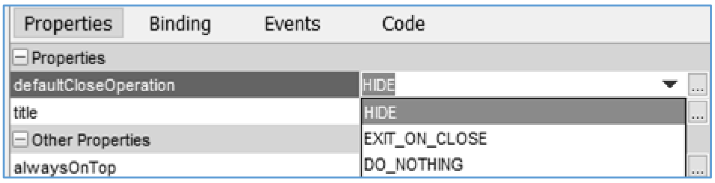
Begin by defining the graphics font style and size which we will use, and produce a fawn colour to use for the linked list data items. We will then draw a box to contain the ***start pointer***.

```
    private void drawDiagram()
    {
        loadData();

        Font sanSerifFont = new Font("SanSerif", Font.PLAIN, 12);
        Graphics2D g = (Graphics2D) pnlDiagram.getGraphics();
        g.setFont(sanSerifFont);
        int red=0xFF;
        int green=0xE4;
        int blue=0xB5;
        Color fawn = new Color(red,green,blue);
        String s="Start";
        int x=100;
        int y=60;
        g.setColor(Color.black);
        g.drawRect(x,y,120,20);
        g.drawLine(x+44, y, x+44, y+20);
        g.drawString(s,x+8,y+14);
        s="pointer "+startPointer;
        g.drawString(s,x+58,y+14);
    }
```

Run the program, then use the '***Open***' menu option to load the linked list. The data should be displayed in the table. Click the '***Display linked list***' menu option to open the graphics window. Move the mouse onto the white panel. A box containing the ***start pointer value*** should appear, as shown below.



Click the cross at the top right to close the graphics window. We would like to return to the main program window so that further names can be added to the table, but unfortunately the whole program closes! Go to the ***linkedListDiagram.java*** page to correct this. Click the ***Design*** tab to move to the form layout view, then click on an area outside the panel to select the form. Go to the ***Properties*** window and set the ***defaultCloseOperation*** to ***HIDE*** by selecting from the drop down list.



defaultCloseOperation HIDE

title

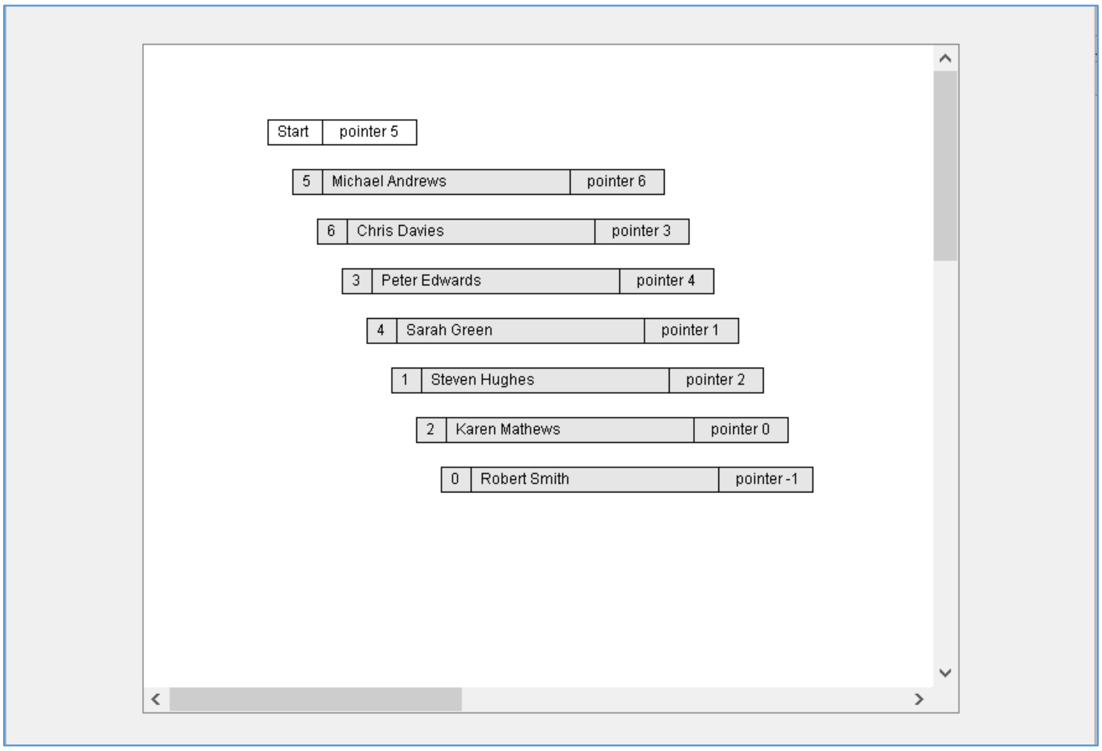
alwaysOnTop

Use the ***Source*** tab to move to the program code view. We will continue work on the ***drawDiagram( )*** method. Add lines of code to draw boxes representing the linked list items.

```
        g.drawString(s,x+8,y+14);
        s="pointer "+startPointer;
        g.drawString(s,x+58,y+14);

        Boolean finished=false;
        int current=startPointer;
        while(finished==false)
        {
            x=x+20;
            y=y+40;
            g.setColor(fawn);
            g.fillRect(x,y,300,20);
            g.setColor(Color.black);
            g.drawRect(x,y,300,20);
            g.drawLine(x+24, y, x+24, y+20);
            s=String.valueOf(current);
            g.drawString(s,x+8,y+14);
            s=forename[current]+" "+surname[current];
            g.drawString(s,x+32,y+14);
            s="pointer "+pointer[current];
            g.drawString(s,x+238,y+14);
            g.drawLine(x+224, y, x+224, y+20);
            current=pointer[current];
            if (current<0)
            {
               finished=true;
            }
        }
    }
```

Notice that the program uses the ***current*** variable to record the array position of the data item which is currently being added to the diagram. The pointer for each data item links to the next item to be displayed, until the end of the list is reached.

Run the program and use the '***Open***' option to load the data into the table. Select the '***Display linked list***' option to open the graphics window, then move the mouse onto the white panel. The sequence of ***names*** should be shown, along with the ***location*** of each record and the ***pointer*** to the next record in the sequence. The first record is accessed by the ***start pointer***, and the last record has a pointer value of **-1** to indicate the ***end of the list***.

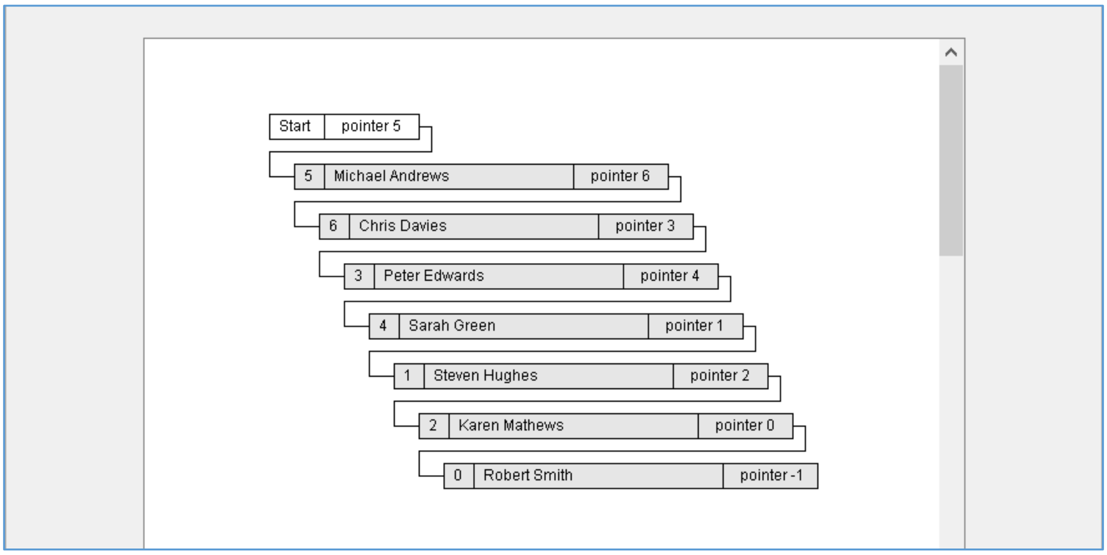


Start | pointer 5
5 | Michael Andrews | pointer 6
6 | Chris Davies | pointer 3
3 | Peter Edwards | pointer 4
4 | Sarah Green | pointer 1
1 | Steven Hughes | pointer 2
2 | Karen Mathews | pointer 0
0 | Robert Smith | pointer -1

Close the graphics window and exit from the main program to return to the NetBeans editing screen.

We will add a final improvement to the diagram, which is to draw lines to link the sequence of records. Locate the ***drawDiagram( )*** method and add lines of code as shown on the next page.

When you have added the extra lines of code then run the program, load the data file, and select the '***Display linked list***' option. Lines linking the records should now be added.

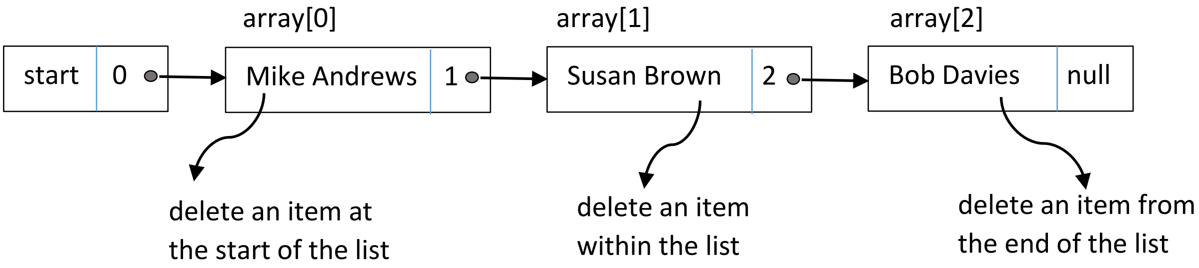


Start | pointer 5
5 | Michael Andrews | pointer 6
6 | Chris Davies | pointer 3
3 | Peter Edwards | pointer 4
4 | Sarah Green | pointer 1
1 | Steven Hughes | pointer 2
2 | Karen Mathews | pointer 0
0 | Robert Smith | pointer -1

```
private void drawDiagram()
{
    loadData();
    Font sanSerifFont = new Font("SanSerif", Font.PLAIN, 12);
    Graphics2D g = (Graphics2D) pnlDiagram.getGraphics();
    g.setFont(sanSerifFont);
    int red=0xFF;
    int green=0xE4;
    int blue=0xB5;
    Color fawn = new Color(red,green,blue);
    String s="Start";
    int x=100;
    int y=60;
    g.setColor(Color.black);
    g.drawRect(x,y,120,20);
    g.drawLine(x+44, y, x+44, y+20);
    g.drawString(s,x+8,y+14);
    s="pointer "+startPointer;
    g.drawString(s,x+58,y+14);

    g.drawLine(x+120, y+10, x+130, y+10);
    g.drawLine(x+130, y+10, x+130, y+30);
    g.drawLine(x, y+30, x+130, y+30);
    g.drawLine(x, y+30, x, y+50);
    g.drawLine(x, y+50, x+20, y+50);

    Boolean finished=false;
    int current=startPointer;
    while(finished==false)
    {
        x=x+20;
        y=y+40;
        g.setColor(fawn);
        g.fillRect(x,y,300,20);
        g.setColor(Color.black);
        g.drawRect(x,y,300,20);
        g.drawLine(x+24, y, x+24, y+20);
        s=String.valueOf(current);
        g.drawString(s,x+8,y+14);
        s=forename[current]+" "+surname[current];
        g.drawString(s,x+32,y+14);
        s="pointer "+pointer[current];
        g.drawString(s,x+238,y+14);
        g.drawLine(x+224, y, x+224, y+20);
        current=pointer[current];
        if (current<0)
        {
           finished=true;
        }

        else
        {
            g.drawLine(x+300, y+10, x+310, y+10);
            g.drawLine(x+310, y+10, x+310, y+30);
            g.drawLine(x, y+30, x+310, y+30);
            g.drawLine(x, y+30, x, y+50);
            g.drawLine(x, y+50, x+20, y+50);
        }

    }
}
```

After testing the program, close the windows and return to the NetBeans editing screen.

To complete the linked list project, we should provide procedures for removing items for the linked list. As in the case of adding items, there are three different cases which must be considered:


array[0]
array[1]
array[2]
start
0
Mike Andrews
1
Susan Brown
2
Bob Davies
null
delete an item at the start of the list
delete an item within the list
delete an item from the end of the list

Each of these options will need to be considered separately, as different combinations of pointers are involved.

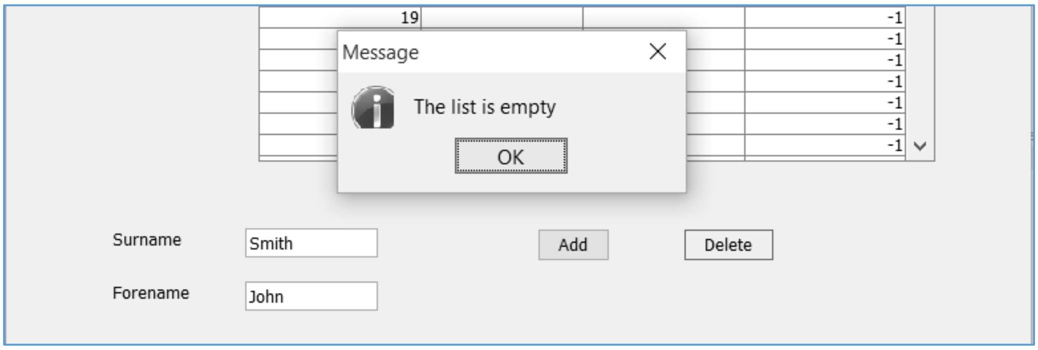
Use the tab at the top of the editing screen to open the ***linkedList.java*** page. Click the ***Design*** tab to move to the form layout view, then double-click the '***Delete***' button to create a method. Add code which will access the text fields to obtain the name of the person to be deleted, carry out a presence check on the data, then call a ***remove( )*** method. Begin the ***remove( )*** method underneath.

```
private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
    String surnameEntered=txtSurname.getText();
    String forenameEntered=txtForename.getText();
    if(surnameEntered.length()>0 && forenameEntered.length()>0)
    {
       remove(surnameEntered,forenameEntered);
    }
}
private void remove(String surnameEntered, String forenameEntered)
{

}
```

Continue the ***remove( )*** method by adding definitions for the variables which will be needed by the program. We will then add error trapping to warn the user if they attempt to remove an item from a list which is already empty.

```
private void remove(String surnameEntered, String forenameEntered)
{
    int previous=startPointer;
    Boolean finished;
    String nameEntered=surnameEntered+" "+forenameEntered;
    String name;
    Boolean found=false;
    int current=startPointer;
    if (surname[current]==null)
    {
       JOptionPane.showMessageDialog(linkedList.this, "The list is empty");
    }
 }
```

Run the program, but do not add or load data. Enter a name in the text fields, then click the '***Delete***' button. Check that an error message is given.



The list is empty

Surname Smith

Forename John

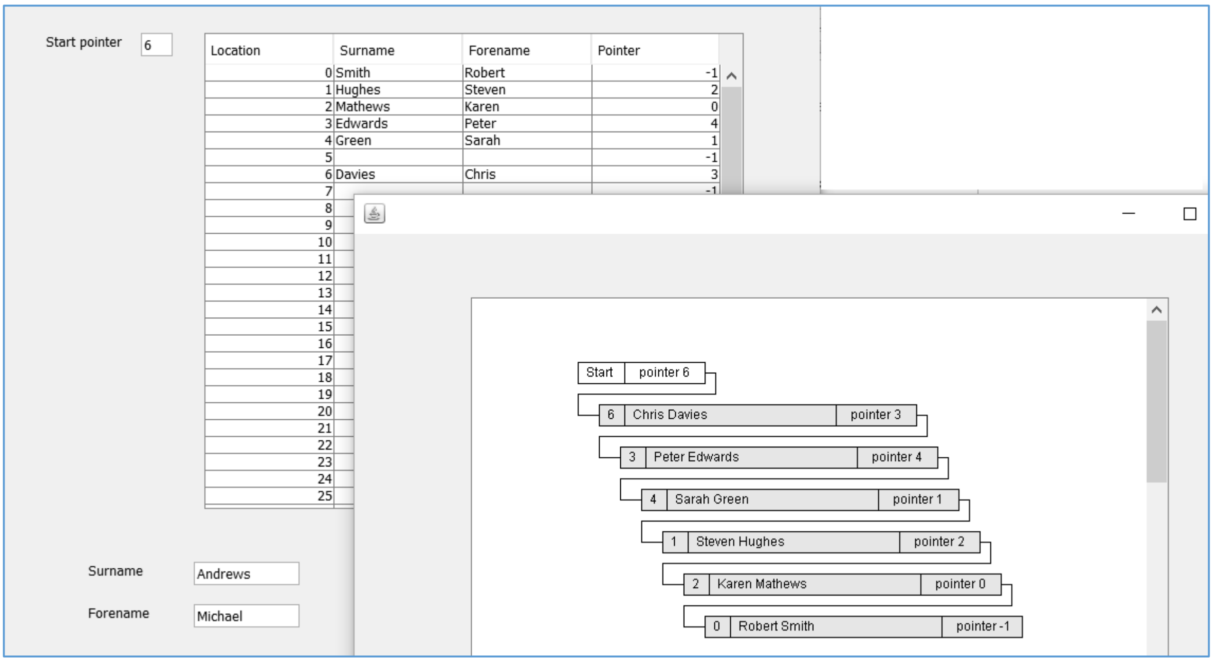
Close the program window and return to the code editing screen.

We will now consider the case where the first item in the list is deleted. Add lines of code to the ***remove( )*** method, then insert a ***removeStart( )*** method underneath.

```
        if (surname[current]==null)
        {
           JOptionPane.showMessageDialog(linkedList.this, "The list is empty");
        }
        else
        {
           name=surname[current]+" "+forename[current];
           if (name.compareTo(nameEntered)==0)
           {
               removeStart(current);
               found=true;
           }
           if (found==false)
           {
               JOptionPane.showMessageDialog(linkedList.this,"Name not found");
           }
           else
           {
               displayTable();
           }
        }
    }

    private void removeStart(int current)
    {
        surname[current]=null;
        forename[current]=null;
        if (pointer[current]>=0)
        {
            startPointer=pointer[current];
            pointer[current]=-1;
        }
    }
```

The ***removeStart( )*** method clears the deleted name from the arrays, and changes the ***start pointer*** so that the next list item becomes the new start of the linked list.

Run the program and use the '***Open***' menu option to load the data file. Identify the first name in the linked list. Enter this name in the text fields, then click the '***Delete***' button. Check that the name has been deleted from the table, and the ***start pointer*** has been updated. Open the linked list diagram and check that this is displaying the start of the list correctly.



Start pointer 6

Location	Surname	Forename	Pointer
0	Smith	Robert	-1
1	Hughes	Steven	2
2	Mathews	Karen	0
3	Edwards	Peter	4
4	Green	Sarah	1
5			-1
6	Davies	Chris	3
7			-1
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Surname Andrews

Forename Michael

Start | pointer 6

6 | Chris Davies | pointer 3

3 | Peter Edwards | pointer 4

4 | Sarah Green | pointer 1

1 | Steven Hughes | pointer 2

2 | Karen Mathews | pointer 0

0 | Robert Smith | pointer -1

Close the program windows and return to the code editing screen. We will now consider the case where the last item in the list is deleted. Add lines of code to the ***remove( )*** method.

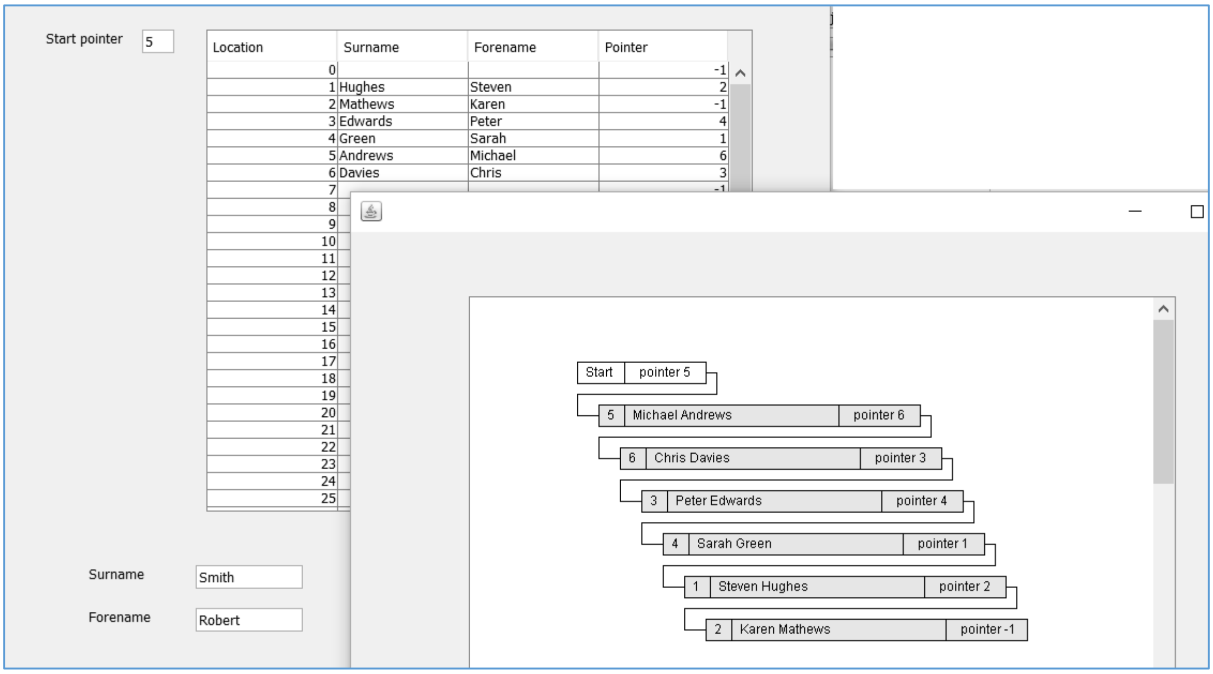
```
      if (name.compareTo(nameEntered)==0)
      {
          removeStart(current);
          found=true;
      }
      else
      {
          finished=false;
          while(finished==false)
          {
              if(pointer[current]<0)
              {
                  finished=true;
                  name=surname[current]+" "+forename[current];
                  if (name.compareTo(nameEntered)==0)
                  {
                     removeEnd(current,previous);
                     found=true;
                  }
              }
              previous=current;
              current=pointer[current];
          }
       }
       if (found==false)
       {
           JOptionPane.showMessageDialog(linkedList.this, "Name not found");
       }
```

Add a ***removeEnd( )*** method below the ***remove( )*** method.

```
private void removeEnd(int current, int previous)
{
    surname[current]=null;
    forename[current]=null;
    pointer[current]=-1;
    pointer[previous]=-1;
}
```

The ***removeEnd( )*** method clears the deleted name from the arrays and resets the pointer from the previous record to **-1**, so that the previous record becomes the new final record in the linked list.

Run the program and use the '***Open***' menu option to load the data file. Identify the last name in the linked list. Enter this name in the text fields, then click the '***Delete***' button. Check that the name has been deleted from the table, and the pointer from the previous record has been reset to -1. Open the linked list diagram and check that this is displaying the end of the list correctly.



Start pointer 5

Location	Surname	Forename	Pointer
0			-1
1	Hughes	Steven	2
2	Mathews	Karen	-1
3	Edwards	Peter	4
4	Green	Sarah	1
5	Andrews	Michael	6
6	Davies	Chris	3
7			-1
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Surname Smith

Forename Robert

Start pointer 5
5 Michael Andrews pointer 6
6 Chris Davies pointer 3
3 Peter Edwards pointer 4
4 Sarah Green pointer 1
1 Steven Hughes pointer 2
2 Karen Mathews pointer -1

Close the program windows and return to the code editing screen. The final case we have to consider is an item being deleted from ***within*** the linked list sequence.

Add further lines of code to the ***remove( )*** method, as shown on the next page, and insert a ***removeMid( )*** method underneath.

The ***removeMid( )*** method clears the deleted name from the arrays. The pointer of the previous record is then reset to the location of the record which is next ahead in the sequence, missing out the deleted record.

```
           while(finished==false)
           {
               if(pointer[current]<0)
               {
                   finished=true;
                   name=surname[current]+" "+forename[current];
                   if (name.compareTo(nameEntered)==0)
                   {
                       removeEnd(current,previous);
                       found=true;
                   }
               }
               else
               {
                   name=surname[current]+" "+forename[current];
                   System.out.println("Name = "+name);
                   System.out.println("NameEntered = "+nameEntered);
                   if (name.compareTo(nameEntered)==0)
                   {
                       finished=true;
                       removeMid(current,previous);
                       found=true;
                       finished=true;
                   }
               }
              previous=current;
              current=pointer[current];
           }
        }
        if (found==false)
        {
            JOptionPane.showMessageDialog(linkedList.this, "Name not found");
        }
        else
        {
              displayTable();
        }
    }
}
private void removeMid(int current, int previous)
{
    surname[current]=null;
    forename[current]=null;
    pointer[previous]=pointer[current];
    pointer[current]=-1;
}
```

Run the completed program. Check that:

- Names can be added at any position in the linked list.
- Names can be deleted from any position in the list.
- A linked list can be saved on disc, then reloaded and displayed in the table.
- The linked list is shown in correct alphabetical sequence in the diagram.