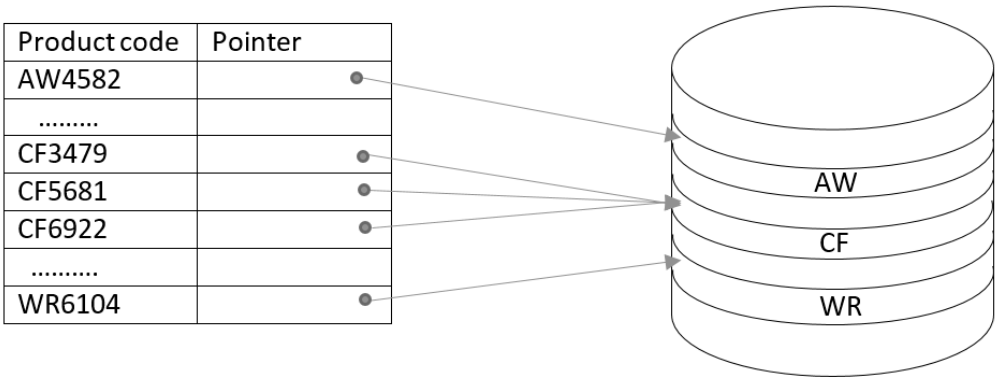
# 17 Indexed sequential files

In the previous chapter, we looked at the Random Access File method for accessing records in large disc-based systems. Another approach to providing fast access to records on disc is to use an ***index***, similar to a telephone directory, which lists the records in ***alphabetical*** or ***numerical order***. The index then specifies the memory location on disc where each particular record can be found. This system has the advantage that the index may be small enough to be held in the ***electronic main memory*** of the computer. Searching the index should be ***very fast***, and only one slow disc access operation will be needed to load the required record.

The index will normally be a list of the ***key field*** values of the records in sorted order, providing a pointer to the storage location of each record on disc. A possible approach is shown in this example:




Pointers may lead to single records. However, it is sometimes more efficient for records to be grouped together in blocks on disc, as in the case of the three records beginning ***CF***. The records can be loaded from disc ***as a block***, then placed in a ***buffer area*** in the electronic memory where the required record will be extracted. To speed up this operation, the records within any block can be stored in ***sequential order***, so that a fast ***binary search*** can be carried out:

| | | | | | | |
|---|---|---|---|---|---|---|
| CF1344 | ..... | CF3479 | CF5681 | CF6922 | ..... | CF9235 |

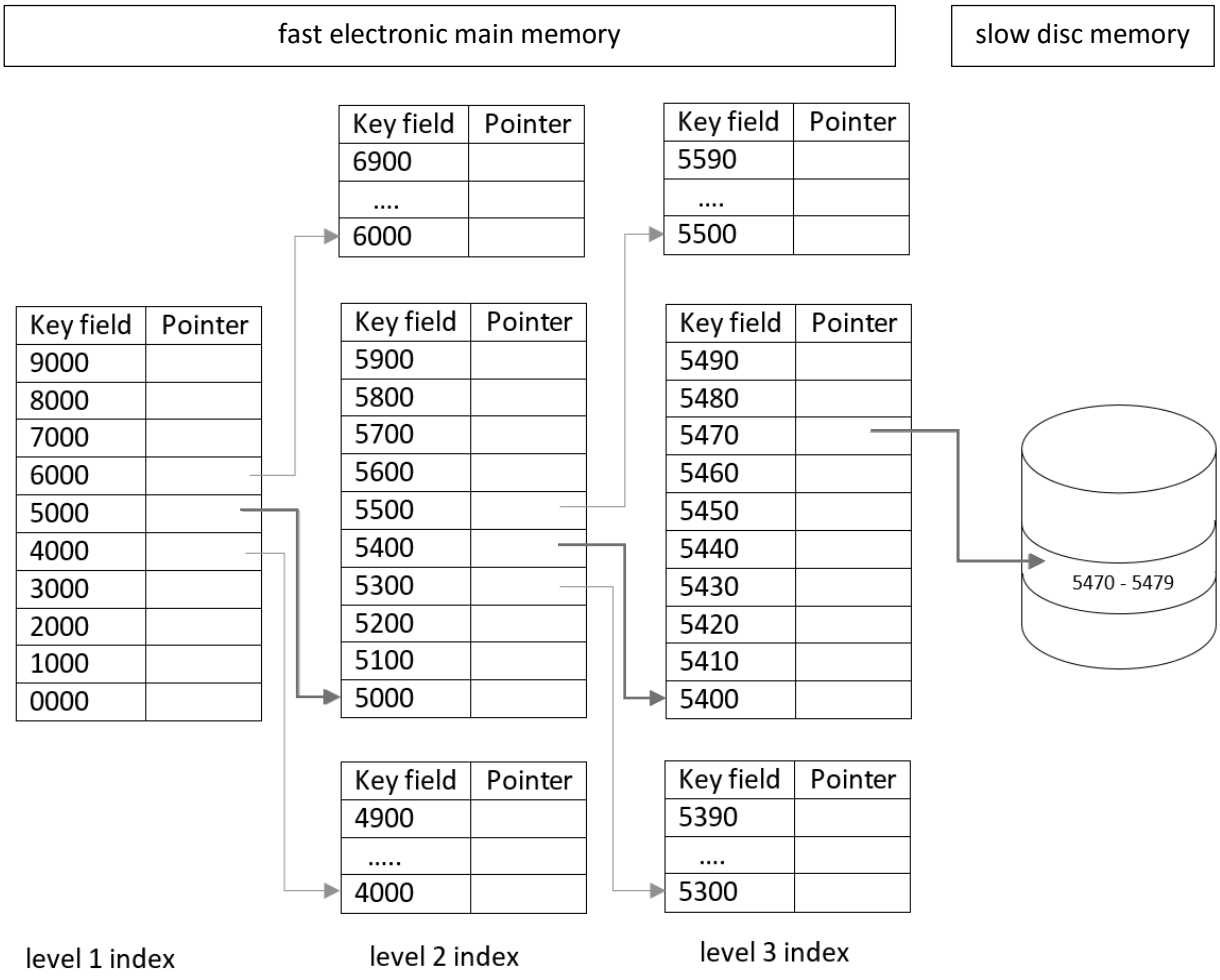
The combination of an index, plus groups of records stored on disc in sequential order, gives the name ***indexed sequential file*** for this type of storage system.

This system would work well for moderate numbers of records. However, if the number of records becomes very large, then the process of searching the index itself could become slow. An improvement is to reduce the number of search operations by providing several ***levels of index***.

Suppose that customer records are numbered in the range 0000 to 9999. A ***three-level index system*** could be set up, as shown in the diagram on the next page. Each Level 1 index block would give access to ten Level 2 index blocks, and each Level 2 index block would in turn give access to ten Level 3 index blocks.

The record for ***customer 5473*** is required. The search takes places in stages:

- The ***level 1 index*** is searched first, and this leads to the level 2 index block for records in the range 5000 – 5999.
- The ***level 2 index*** block is searched next, and this leads to the level 3 index block for records in the range 5400 – 5499.
- The ***level 3 index*** block is searched, and this gives the location of the block of records 5470 -5479 in the ***disc file***.
- The block of records is loaded, and the required record ***5473*** is extracted.




For our next project, we will set up a ***three level indexed sequential file*** system for a travel agent to handle records of holidays available. We will assume that the travel agent has a large number of different dates, locations and accommodation choices available to customers, and each unique combination is given a ***HolidayID*** number in the range ***0000 – 9999***.

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

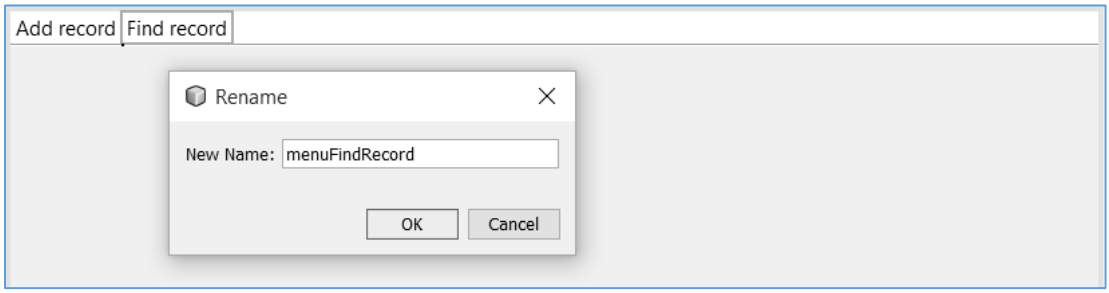
Return to the NetBeans editing page. Right-click on the ***indexedFile*** project, and select ***New*** / ***JFrame Form***. Give the ***Class Name*** as ***indexedFile,*** and the ***Package*** as ***indexedFilePackage***.

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

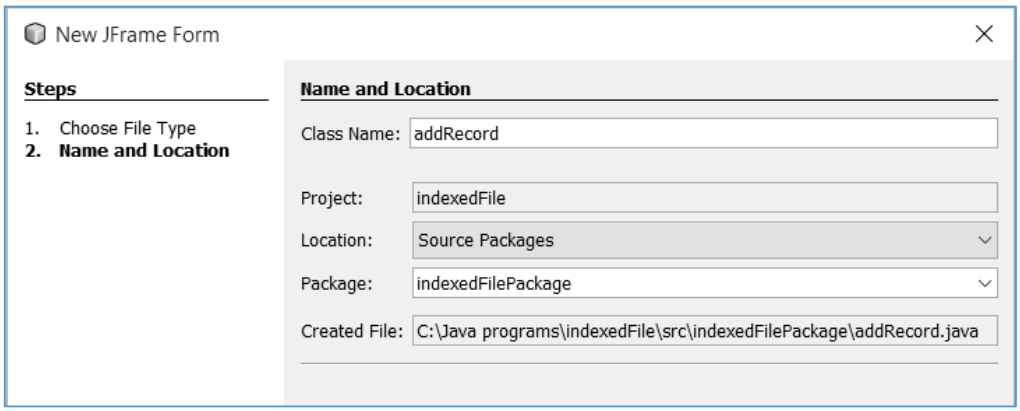
- Right-click on the ***form***, and select ***Set layout*** / ***Absolute layout***.
- Go to the ***Properties*** window on the bottom right of the screen and click the ***Code*** tab. Select the option: ***Form Size Policy*** / ***Generate pack()*** / ***Generate Resize code***.
- Click the Source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

Select a Menu Bar component from the palette, then drag and drop this on the form. Right-click on each of the menu items and change the captions to '***Add record***' and '***Find record***'. Rename the menu items as ***menuAddRecord*** and ***menuFindRecord***.




We will now create the forms which will be linked to these menu items. Go to the Projects window and right-click on the ***indexedFilePackage*** folder. Select ***New*** / ***JFrame Form***. Set the Class Name to '***addRecord***', leaving the Package name as ***indexedFilePackage***.

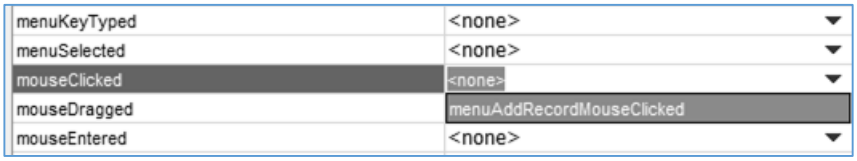



Click the Finish button to open the new form.

- 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***.
- Set the ***defaultCloseOperation*** property to '***HIDE***'.

Repeat the steps above to create another new form with the Class Name '***findRecord***', leaving the Package name as ***indexedFilePackage***. When the form opens, select ***Set layout*** / ***Absolute layout***, ***Form Size Policy*** / ***Generate pack()*** / ***Generate Resize code*** and ***defaultCloseOperation*** / '***HIDE***'.

Use the tab above the editing screen to return to the ***indexedFile.java*** page. Select the '***Add record***' menu item, then go the Properties window and click the ***Events*** tab. Locate the ***mouseClicked*** event, then accept ***menuAddRecordMouseClicked*** from the drop down list.



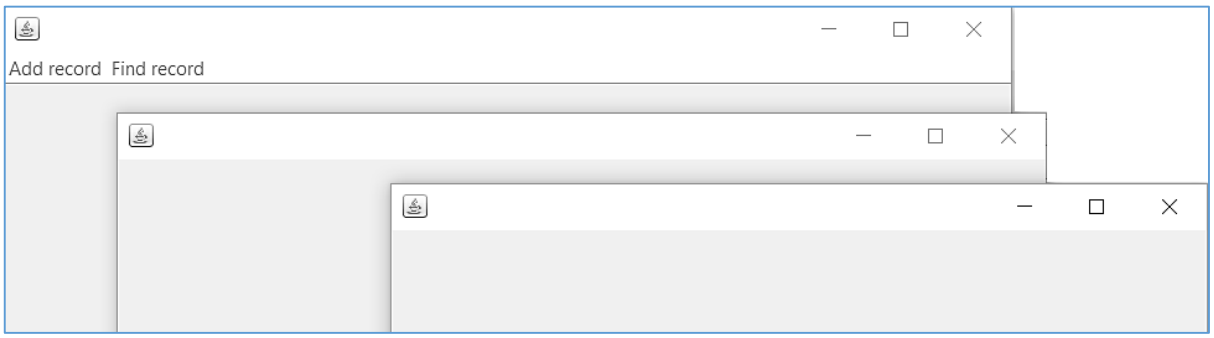

Add a line of code to the ***mouseClicked( )*** method to open the ***addRecord*** form.

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

      new addRecord().setVisible(true);
   }
```

Use the Design tab to return to the form layout view. Select the '***Find record***' menu option. Go to the ***Events*** list and select ***mouseClicked*** / ***menuFindRecordMouseClicked***. Add a line of code to the ***mouseClicked( )*** method to open the ***findRecord*** form.

```
   private void menuFindRecordMouseClicked(java.awt.event.MouseEvent evt) {
      new findRecord().setVisible(true);
   }
```

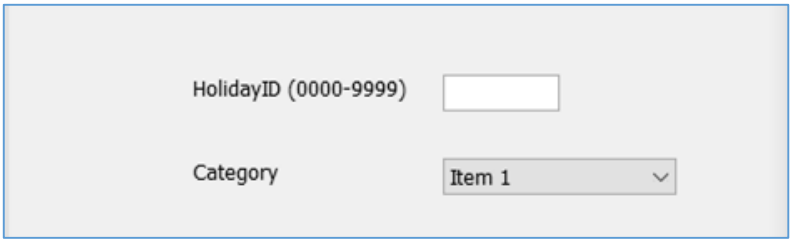
Run the program. Check that each of the windows can be opened by clicking the corresponding menu item, and that the windows can be closed again without exiting from the main program.



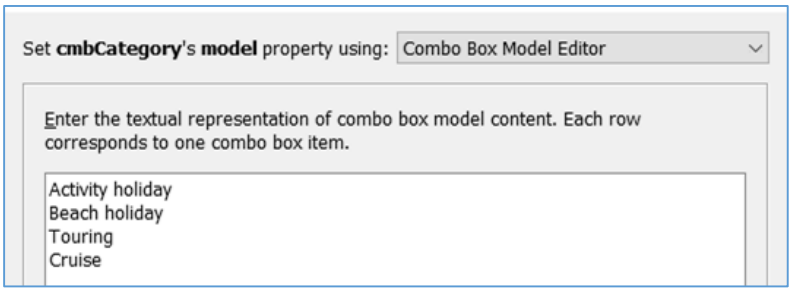

Close the program windows and return to the NetBeans screen. Use the tab above the editing window to move to the ***addRecord.java*** page.

Add components to the form, as shown below:

- A label '***HolidayID (0000 – 9999)***', with a text field alongside. Rename the text field as ***txtHolidayID***.
- A label '***Category***' with a Combo Box alongside. Rename the Combo Box as ***cmbCategory***.

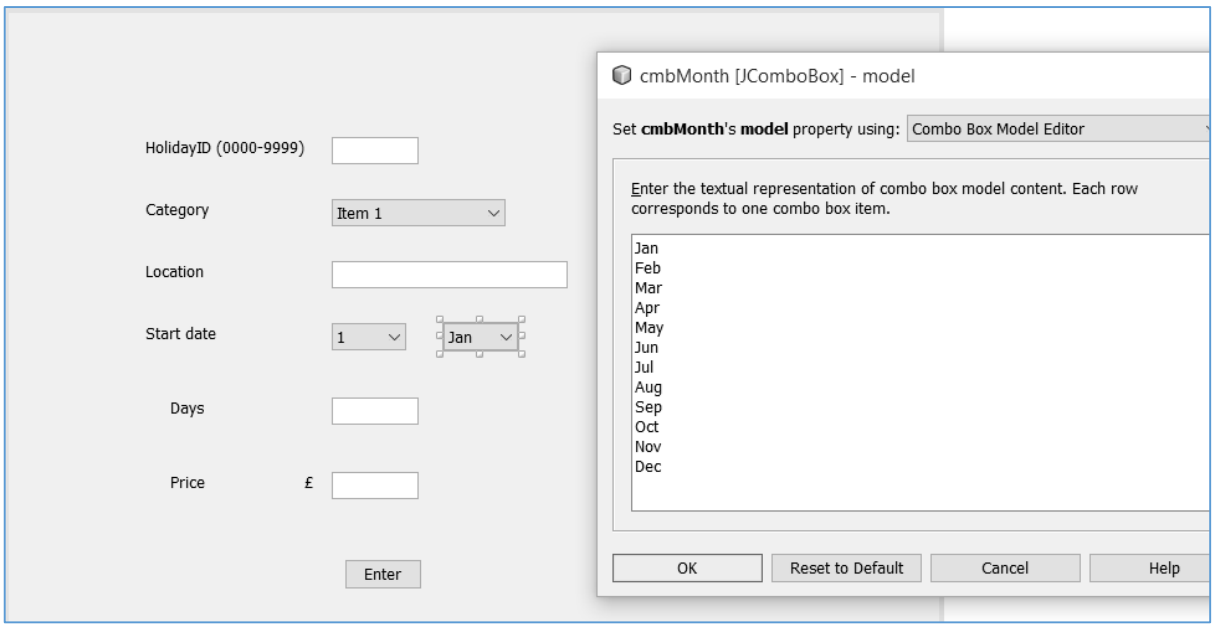



Select the ***Category*** Combo Box, then go to the Properties window and locate the model property. Click the ellipsis (…) symbol at the end of the ***model*** property row to open an editing window. List the categories of holiday: ***Activity holiday***, ***Beach holiday***, ***Touring***, ***Cruise***.




Add further components to the form:

- A label '***Location***', with a text field alongside. Rename the text field as ***txtLocation***.
- A label '***Start date***'. Place two Combo Boxes alongside, renaming these as ***cmbDay*** and ***cmbMonth***. Select each of the Combo Boxes in turn, clicking the ellipsis (…) symbol at the end of the ***model*** property row to open the editing window. Enter the numbers ***1*** to ***31*** as the drop down list options for ***cmbDay***, and the month names ***Jan*** to ***Dec*** for ***cmbMonth*** as shown below.
- A label '***Days***', with a text field alongside. Rename the text field as ***txtDays***.
- A label '***Price***', with a text field alongside. Rename the text field as ***txtPrice***.
- A button with the caption '***Enter***'. Rename the button as ***btnEnter***.



Click the ***Source*** tab to move to the program code page. Go to the start of the program listing and add the Java modules which will be needed for file handling and displaying message boxes.

```
package indexedFilePackage;

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

public class addRecord extends javax.swing.JFrame {

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

Click the ***Design*** tab to move to the form view. Double click the '***Enter***' button to create a method. Add a line of code to call a ***save( )*** method. We will insert the ***save( )*** method immediately below.

Add lines of code to ***save( )*** which will check that the holiday code entered is four digits in length, and contains no incorrect characters. Please note that the lines beginning:

***JOptionPane.showMessageDialog(…***

should be entered as single lines of code with no line break.

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

    save();

}

private void save()
{
    String holidayCode=txtHolidayID.getText();
    holidayCode.trim();
    if (holidayCode.length()!=4)
    {
        JOptionPane.showMessageDialog(addRecord.this,
                                "The holiday code must be four digits");
    }
    else
    {
        try
        {
            int holidayID = Integer.parseInt(holidayCode);
        }
        catch(NumberFormatException e)
        {
            JOptionPane.showMessageDialog(addRecord.this,
                                            "Incorrect number format");

        }
    }
}
```

Run the program, then select the '***Add record***' menu option. Test the error trapping for HolidayID numbers of incorrect length, or containing incorrect characters.

Close the program and return to the NetBeans editing screen.

We will save the holiday data as fixed length records containing six fields. Suitable field sizes can be allocated:

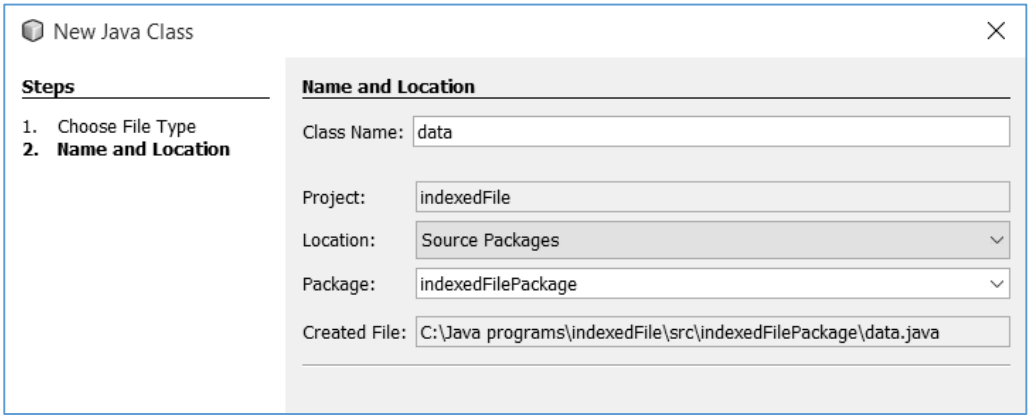
| HolidayID | Category | Location | Start Date | Days | Price |
|---|---|---|---|---|---|
| 10 bytes | 20 bytes | 60 bytes | 10 bytes | 10 bytes | 10 bytes |

Locate the ***save( )*** method. Add lines of code which will collect the holiday information from the input components, set these to the correct fixed field lengths, then compile the fields into a record.

```
private void save()

{
  String holidayCode=txtHolidayID.getText();
  holidayCode.trim();
  if (holidayCode.length()!=4)
  {
    JOptionPane.showMessageDialog(addRecord.this,
                    "The holiday code must be four digits");
  }
  else
  {
     try
     {
       int holidayID = Integer.parseInt(holidayCode);

       String category=cmbCategory.getSelectedItem().toString();
       String location=txtLocation.getText();
       String startDate=cmbDay.getSelectedItem()+" "+cmbMonth.getSelectedItem();
       String days=txtDays.getText();
       String price=txtPrice.getText();

       holidayCode=String.format("%-10s", holidayID);
       category=String.format("%-20s", category);
       location=String.format("%-60s", location);
       startDate=String.format("%-10s", startDate);
       days=String.format("%-10s", days);
       price=String.format("%-10s", price);
       String holidayRecord= holidayCode+category+location+startDate+days+price;
     }
     catch(NumberFormatException e)
     {
```

Two files will be needed to operate the indexed file system: one to hold the ***index***, and the other to hold the actual ***holiday records***. The file names should be available to all parts of the program, so it is best to define these as ***global variables*** in a ***data class***.

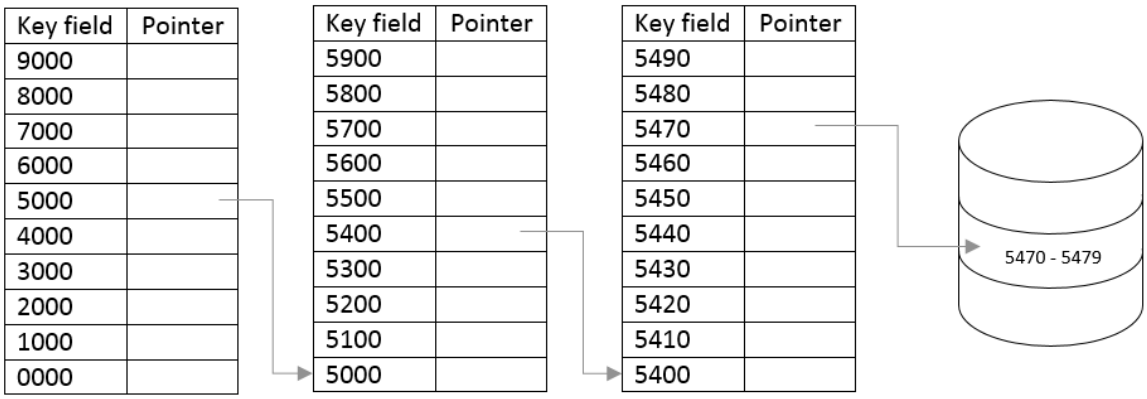
Go to the Projects window at the top left of the screen, and right-click on the ***indexedFilePackage*** folder. Select ***New*** / ***Java Class***. Set the Class Name as '***data***', leaving the Package name as ***indexedFilePackage***, as shown below.




Click the ***Finish*** button to open the ***data*** class file. Add names for the index and holiday data files.

```
package indexedFilePackage;

public class data {

    public static String indexFile = "index.dat";
    public static String holidayFile = "holidays.dat";

}
```

To develop the holiday records system, we will now work through the steps required to store a record with a HolidayID value of ***5473***, as in the example at the start of this chapter.




The indexed file system begins with a ***Level 1 index*** which should contain the entries '***0000, 1000, 2000, … 9000***'. This will be created when the program is first run.

Use the tab above the editing screen to move to the ***indexedFile.java*** page. Click the ***Source*** tab to move to the program code view. We will add lines of code which:

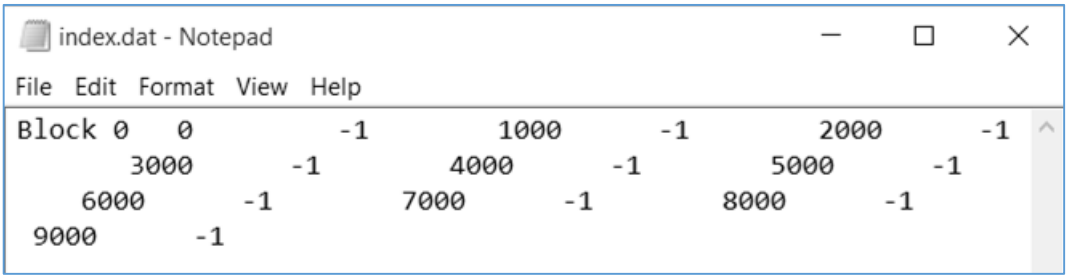
- Include Java modules needed for file operations.
- Check whether an index file already exists. If not, a method is called to create the index.
- Begin the ***createIndexFile( )*** method immediately underneath.

Add a loop to the the ***createIndexFile( )*** method which will:

- Add a caption '***Block 0***' to the file.
- Generate numbers at intervals of ***1000***, and insert these into the index.
- Add a ***link value of -1*** for each location, to show that we have not yet set up any connections to level 2 index blocks.

```
package indexedFilePackage;

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

public class indexedFile extends javax.swing.JFrame {

    public indexedFile() {
        initComponents();

        File f = new File(data.indexFile);
        if(f.exists()==false)
        {
            createIndexFile();
        }
    }

    private void createIndexFile()
    {
       try (RandomAccessFile file = new RandomAccessFile(data.indexFile, "rw"))
       {
           String blockNumber="Block 0";
           blockNumber=String.format("%-10s", blockNumber);
           file.write(blockNumber.getBytes());
           for (int i=0; i<10; i++)
           {
              String locationNumber=String.format("%-10s", i*1000);
              String link=String.format("%-10s", "-1");
              String indexRecord=locationNumber+link;
              int position=i*20+10;
              file.seek(position);
              file.write(indexRecord.getBytes());
           }
           file.close();
       }
       catch(IOException e)
       {
             JOptionPane.showMessageDialog(indexedFile.this, "File error");
       }
    }
```

Run the program. Use Windows Explorer to locate the ***index.dat*** file in the ***indexedFile*** project folder. Open ***index.dat*** with a text editing application such as Notepad. Check that the sequence of file locations has been created correctly, with a ***-1 pointer*** value for each.



For a very large indexed file application, there could be many ***Level 2*** index blocks, and many more ***Level 3*** index blocks. Even in our relatively small project, there are potentially 111 index bocks. Rather than create all these index blocks initially, the program will only add index blocks when they are actually required by the records in the system.

Close the program window and return to NetBeans. Use the tab above the editing screen to open the ***addRecord.java*** page. Locate the ***save( )*** method.

Once a holiday record has been entered, we will call a method to search the current index blocks for the ***holidayID*** value which has been entered. If the required index blocks do not yet exist, then they will be created. For our example record ***5473***, a ***level 2 index block*** will be needed with entries from ***5000*** to ***5900***.

| Key field | Pointer |
|---|---|
| 9000 | |
| 8000 | |
| 7000 | |
| 6000 | |
| 5000 | |
| 4000 | |
| 3000 | |
| 2000 | |
| 1000 | |
| 0000 | |

| Key field | Pointer |
|---|---|
| 5900 | |
| 5800 | |
| 5700 | |
| 5600 | |
| 5500 | |
| 5400 | |
| 5300 | |
| 5200 | |
| 5100 | |
| 5000 | |

Add a line of code to call a ***searchIndex( )*** method

```
        startDate=String.format("%-10s", startDate);
        days=String.format("%-10s", days);
        price=String.format("%-10s", price);
        String holidayRecord= holidayCode+category+location+startDate+days+price;

        searchIndex(holidayID);

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

Insert the ***searchIndex( )*** method immediately underneath the ***save( )*** method. This begins with a search of the ***Level 1 index***.

- The Level 1 table entry for the ***HolidayID*** value is calculated. Using our example, the relevant Level 1 index entry for HolidayID ***5473*** is ***5000***.
- We then use a method ***getPointer( )*** to return the link value corresponding to this location.

```
private int searchIndex(int holidayID)
{
    int dataBlock = -1;
    int level1=(holidayID /1000)*1000;
    int indexBlock=0;
    int pointer=getPointer(indexBlock,level1);
    return dataBlock;
}
```

Begin the ***getPointer( )*** method immediately below the ***searchIndex( )*** method. This will open the index file and collect the Level 1 index block. The method then checks each of the entries in the index block until the required location number is found, and returns the corresponding pointer value.

```
private int getPointer(int indexBlock, int searchValue)
{
    int pointer=0;
    byte[] bytes=null;
    try (RandomAccessFile file = new RandomAccessFile(data.indexFile, "r"))
    {
        bytes = new byte[210];
        file.seek(indexBlock*210);
        file.read(bytes);
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(addRecord.this, "File error");
    }
    String s = new String(bytes);
    int location;
    String w,x;
    s=s.substring(10);
    int i=0;
    boolean found=false;
    while(found==false)
    {
         w=s.substring(0,10);s=s.substring(10);
         x=s.substring(0,10);s=s.substring(10);
         location=Integer.parseInt(w.trim());
         pointer=Integer.parseInt(x.trim());
         if (location==searchValue)
         {
            found=true;
         }
         i++;
    }
    return pointer;
}
```

If we were to enter a test record using ***HolidayID value 5473***, the ***Level 1 index*** would be searched for ***location 5000***, then a ***pointer value of -1*** would be returned to indicate that there is no ***Level 2 index block*** yet. We will now arrange for the program to create the ***level 2 index block***.

Locate the ***searchIndex( )*** method, then add lines of code which check for a pointer value less than zero. If a negative pointer value is found, a ***createIndexBlock()***method will be called.

Insert the ***createIndexBlock( )*** method immediately below the ***searchIndex( )*** method.

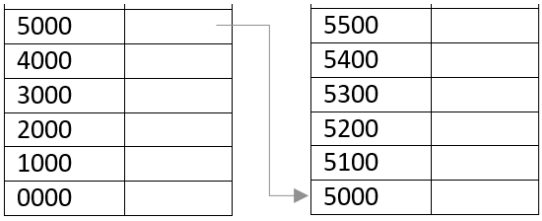
```
private int searchIndex(int holidayID)
{
    int dataBlock = -1;
    int level1=(holidayID /1000)*1000;
    int indexBlock=0;
    int pointer=getPointer(indexBlock,level1);

    if (pointer<0)
    {
        pointer=createIndexBlock(level1, indexBlock, 2);
    }

    return dataBlock;
}
```

```
private int createIndexBlock(int base, int previous, int indexLevel)
{
    int blockCount=0;
    try (RandomAccessFile file = new RandomAccessFile(data.indexFile, "rw"))
    {
        blockCount= (int) (file.length() /210);
        String blockNumber="Block "+blockCount;
        blockNumber=String.format("%-10s", blockNumber);
        int startPosition=blockCount*210;
        file.seek(startPosition);
        file.write(blockNumber.getBytes());
        int multiplier;
        if (indexLevel==2)
        {
            multiplier=100;
        }
        else
        {
            multiplier=10;
        }
        for (int i=0; i<10; i++)
        {
           String locationNumber=String.format("%-10s",base+ i*multiplier);
           String link=String.format("%-10s", "-1");
           String indexRecord=locationNumber+link;
           int position=startPosition + i*20+10;
           file.seek(position);
           file.write(indexRecord.getBytes());
        }
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(addRecord.this, "File error");
    }
    return blockCount;
}
```

The ***searchIndex( )*** method has carried out various tasks:

- The number of index blocks currently in the index file is found.
- The next available block number is allocated to the index block we are now creating, and this is stored into the file as the first line of the new index block.
- The interval between the index entries is found from the index level:
  ***level 2*** index entries are at intervals of ***100***, such as the sequence 5000, 5100, 5200…
  ***level 3*** index entries are at intervals of ***10***, such as the sequence 5400, 5410, 5420…
- A loop creates the 10 entries in the index, setting all the link pointers to values of -1.

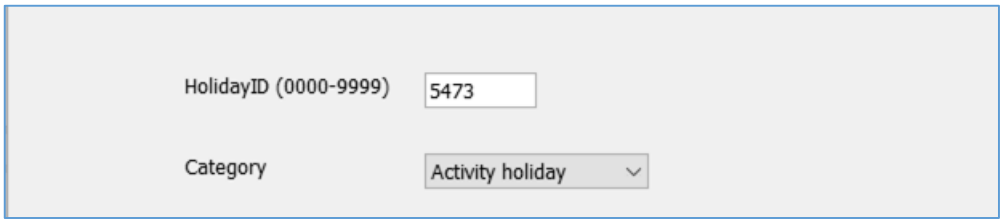
Referring back to the diagram of the index system, you will see that we have one last task to complete.




The level 1 index entry must have a link pointer set to the new level 2 index block. Add lines of code to the ***createIndexBlock( )*** method to access the level 1 index block and reset the link pointer value.

```
    for (int i=0; i<10; i++)
    {
       String locationNumber=String.format("%-10s",base+ i*multiplier);
       String link=String.format("%-10s", "-1");
       String indexRecord=locationNumber+link;
       int position=startPosition + i*20+10;
       file.seek(position);
       file.write(indexRecord.getBytes());
    }

    startPosition=previous*210;
    int row;
    if(indexLevel==2)
    {
        row=base /1000;
    }
    else
    {
        row=base/100;
        row=row % 10;
    }
    String link =Integer.toString(blockCount);
    link=String.format("%-10s", link);
    file.seek(startPosition+row*20 + 20);
    file.write(link.getBytes());

    file.close();
}
 catch(IOException e)
{
      JOptionPane.showMessageDialog(addRecord.this, "File error");
}
```

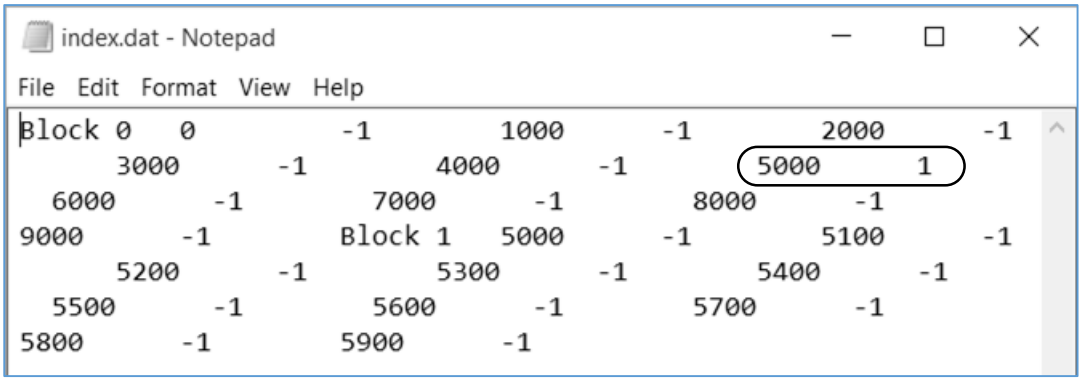
Run the program. Select the '***Add record***' menu option. Enter a ***HolidayID*** value of ***5473***, then click the '***Enter***' button.




Use Windows Explorer to locate the ***index.dat*** file in the ***indexedFile*** project folder. Open the file using a text editing application.

Carefully check the entries in the file:

- We now have two index blocks:
  Block 0 is the level 1 index with location entries from 0 to 9000
  Block 1 is the level 2 index with location entries from 5000 to 5900
- The 5000 location entry in Block 0 has a pointer value of 1, linking it to Block 1.




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

We now need to create a ***level 3*** index block to complete the access sequence for record ***5473***:

| Key field | Pointer |
|---|---|
| 9000 | |
| 8000 | |
| 7000 | |
| 6000 | |
| 5000 | |
| 4000 | |
| 3000 | |
| 2000 | |
| 1000 | |
| 0000 | |

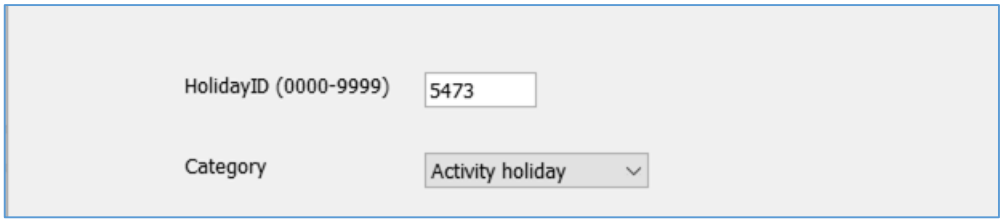
| Key field | Pointer |
|---|---|
| 5900 | |
| 5800 | |
| 5700 | |
| 5600 | |
| 5500 | |
| 5400 | |
| 5300 | |
| 5200 | |
| 5100 | |
| 5000 | |

| Key field | Pointer |
|---|---|
| 5490 | |
| 5480 | |
| 5470 | |
| 5460 | |
| 5450 | |
| 5440 | |
| 5430 | |
| 5420 | |
| 5410 | |
| 5400 | |

We will add code to the ***searchIndex( )*** method to create the level 3 index for the range of locations from 5400 to 5490.

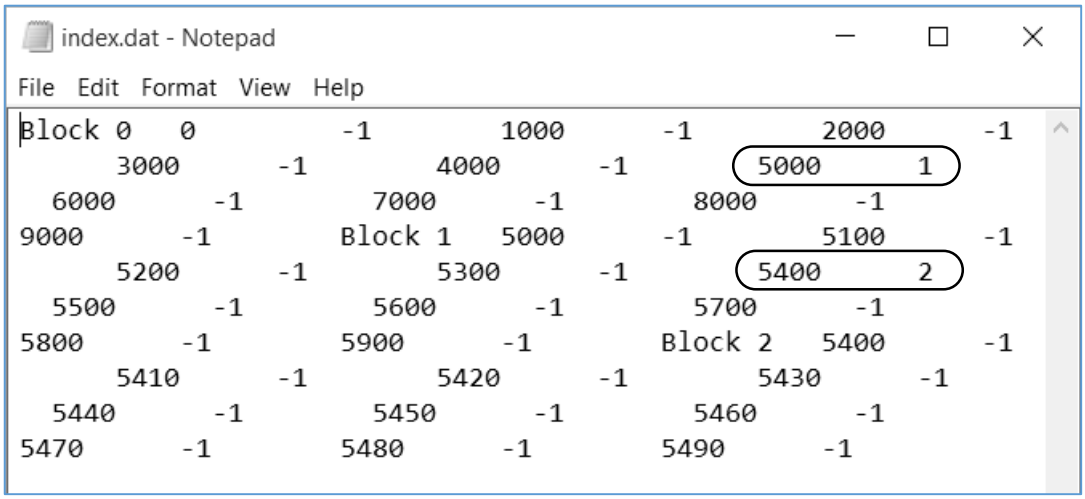
```
private int searchIndex(int holidayID)
{
    int dataBlock = -1;
    int level1=(holidayID /1000)*1000;
    int indexBlock=0;
    int pointer=getPointer(indexBlock,level1);
    if (pointer<0)
    {
        pointer=createIndexBlock(level1, indexBlock, 2);
    }
    int level2=(holidayID /100)*100;
    indexBlock=pointer;
    pointer=getPointer(indexBlock,level2);
    if (pointer<0)
    {
        pointer=createIndexBlock(level2, indexBlock, 3);
    }
    return dataBlock;
}
```

Run the program. Select the '***Add record***' menu option. Enter the ***HolidayID*** value of ***5473***, then click the '***Enter***' button.



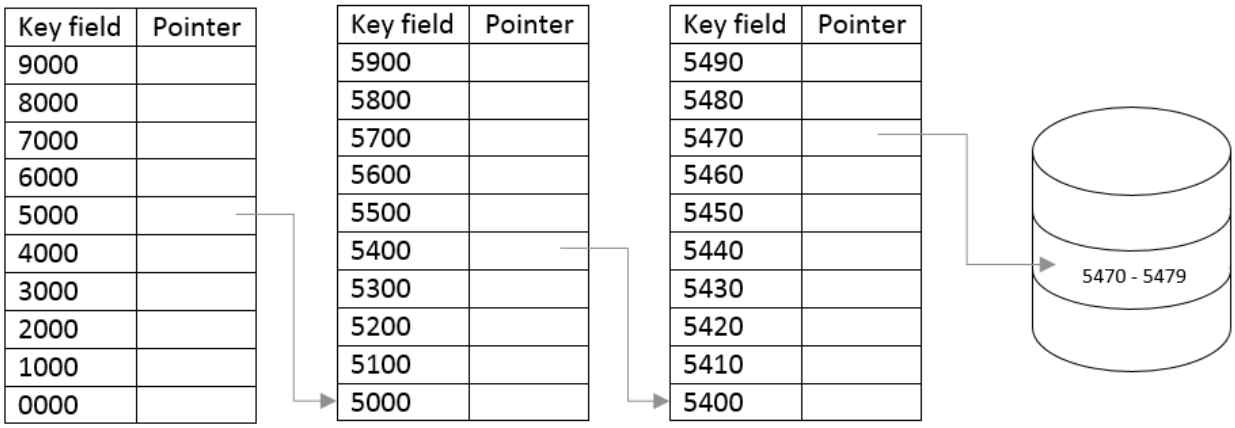

Use Windows Explorer to locate the ***index.dat*** file in the ***indexedFile*** project folder. Open the file using a text editing application. We now have three index blocks, connected by link pointers:

- ***Block 0*** is the level 1 index block with entries from 0 to 9000
- ***Block 1*** is a level 2 index block with entries from 5000 to 5900
- ***Block 2*** is a level 3 index block with entries from 5400 to 5490



We now have a complete three level index sequence, and are ready to save the holiday record in the Holidays data file.

Close the program windows and return to the NetBeans editing screen. Locate the ***searchIndex( )*** method. We will now add lines of code which will check whether a ***data block*** exists yet for the holiday record to be stored. Each data block will have space for a sequence of 10 records. The holiday with ID number ***5473*** will be stored within the data block for records numbered 5470 to 5479.



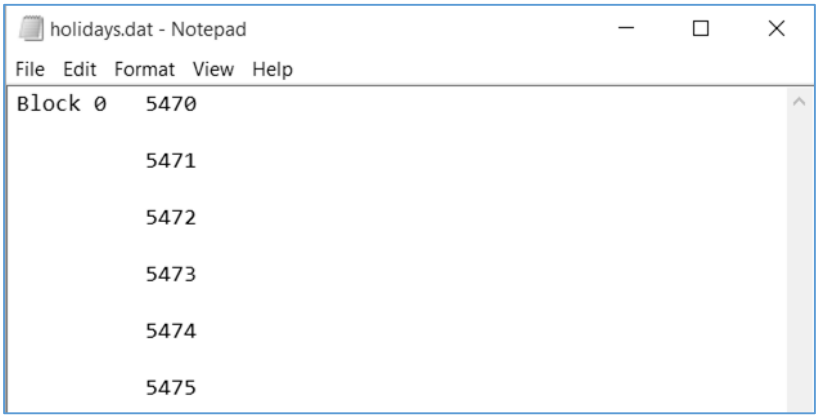

If a data block does not exist yet, it will be created by the ***createDataBlock( )*** method.

```
private int searchIndex(int holidayID)
{
    int dataBlock = -1;
    int level1=(holidayID /1000)*1000;
    int indexBlock=0;
    int pointer=getPointer(indexBlock,level1);
    if (pointer<0)
    {
        pointer=createIndexBlock(level1, indexBlock, 2);
    }
    int level2=(holidayID /100)*100;
    indexBlock=pointer;
    pointer=getPointer(indexBlock,level2);
    if (pointer<0)
    {
        pointer=createIndexBlock(level2, indexBlock, 3);
    }
    int level3=(holidayID /10)*10;
    indexBlock=pointer;
    pointer=getPointer(indexBlock,level3);
    if (pointer<0)
    {
       pointer=createDataBlock(level3, indexBlock);
    }
    dataBlock=pointer;
    return dataBlock;
}
```

Add the ***createDataBlock( )*** method immediately below the ***searchIndex( )*** method.

```
private int createDataBlock(int base, int previous)
{
    int blockCount=-1;

    try (RandomAccessFile file = new RandomAccessFile(data.holidayFile, "rw"))
    {
        blockCount= (int) (file.length() /1210);
        String blockNumber="Block "+blockCount;
        blockNumber=String.format("%-10s", blockNumber);
        int startPosition=blockCount*1210;
        file.seek(startPosition);
        file.write(blockNumber.getBytes());
        for (int i=0; i<10; i++)
        {
           String locationNumber=String.format("%-120s",base+ i);
           String holidayRecord=locationNumber;
           int position=startPosition + i*120+10;
           file.seek(position);
           file.write(holidayRecord.getBytes());
        }
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(addRecord.this, "File error");
    }
    return blockCount;
}
```

Run the program. Select the '***Add record***' menu option. Enter the ***holidayID*** value of ***5473***, then click the '***Enter***' button. Use Windows Explorer to locate the ***holidays.dat*** file in the ***indexedFile*** project folder. Open the file using a text editing application. This file should contain a data block labelled Block 0, with space for inserting holiday records numbered ***5470*** to ***5479***.



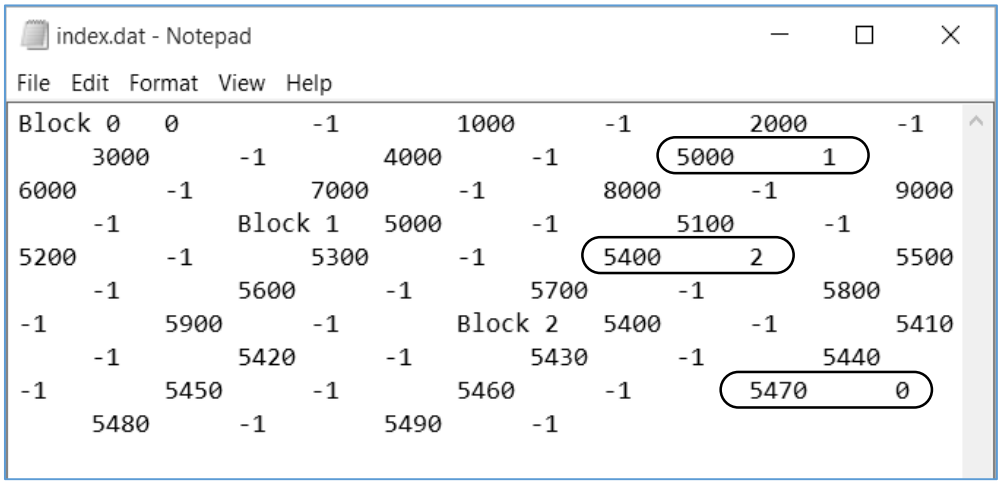

Close the program windows and return to the NetBeans editing screen.

Before continuing, close the text editing application and delete the two data files ***holidays.dat*** and ***index.dat*** from the ***indexedFile*** folder. The program will recreate these files shortly when we carry out the final testing.

Locate the ***createDataBlock( )*** method which you have just written. We will now add lines of code to link the level 3 index block pointer to the data block where our holiday record will be stored.

```
            for (int i=0; i<10; i++)
            {
               String locationNumber=String.format("%-120s",base+ i);
               String holidayRecord=locationNumber;
               int position=startPosition + i*120+10;
               file.seek(position);
               file.write(holidayRecord.getBytes());
            }
            file.close();
        }
        catch(IOException e)
        {
              JOptionPane.showMessageDialog(addRecord.this, "File error");
        }

        try (RandomAccessFile file = new RandomAccessFile(data.indexFile, "rw"))
        {
            int startPosition=previous*210;
            int row;
            row=base/10;
            row=row % 10;
            String link =Integer.toString(blockCount);
            link=String.format("%-10s", link);
            file.seek(startPosition+row*20 + 20);
            file.write(link.getBytes());
            file.close();
        }
         catch(IOException e)
        {
              JOptionPane.showMessageDialog(addRecord.this, "File error");
        }

        return blockCount;
    }
```

Run the program. Select the '***Add record***' menu option. Again enter the ***HolidayID*** value of ***5473***, then click the '***Enter***' button. Use Windows Explorer to locate the ***index.dat*** file in the ***indexedFile*** project folder. Open the file using a text editing application. The link pointer for the entry 5470 in Block 2 should now have been reset to 0, so that data Block 0 will be accessed.



Close the program windows and return to the NetBeans editing screen. The final step is to save the holiday record into the data block at the storage location for HolidayID 5473.

Locate the ***save( )*** method. We are going to change the line '***searchIndex(holidayID)*** ' to read:

***int dataBlock = searchIndex(holidayID)***

then then add a line which calls a ***storeRecord( )*** method.

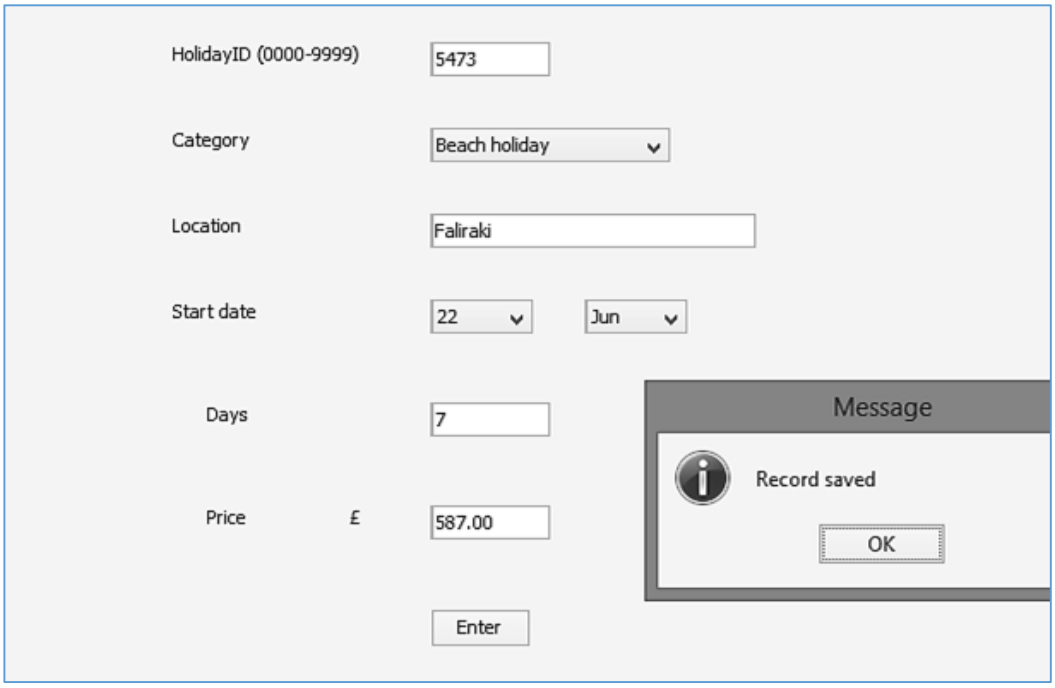
```
try
{
    int holidayID = Integer.parseInt(holidayCode);
    String category=cmbCategory.getSelectedItem().toString();
    String location=txtLocation.getText();
    String startDate=cmbDay.getSelectedItem()+" "+cmbMonth.getSelectedItem();
    String days=txtDays.getText();
    String price=txtPrice.getText();
    holidayCode=String.format("%-10s", holidayID);
    category=String.format("%-20s", category);
    location=String.format("%-60s", location);
    startDate=String.format("%-10s", startDate);
    days=String.format("%-10s", days);
    price=String.format("%-10s", price);
    String holidayRecord= holidayCode+category+location+startDate+days+price;
    int dataBlock=searchIndex(holidayID);
    storeRecord(dataBlock, holidayID, holidayRecord);
    JOptionPane.showMessageDialog(addRecord.this,"Record saved");
}
catch(NumberFormatException e)
{
```

Add the ***storeRecord( )*** method immediately below the ***save( )*** method. This method has three parameters:

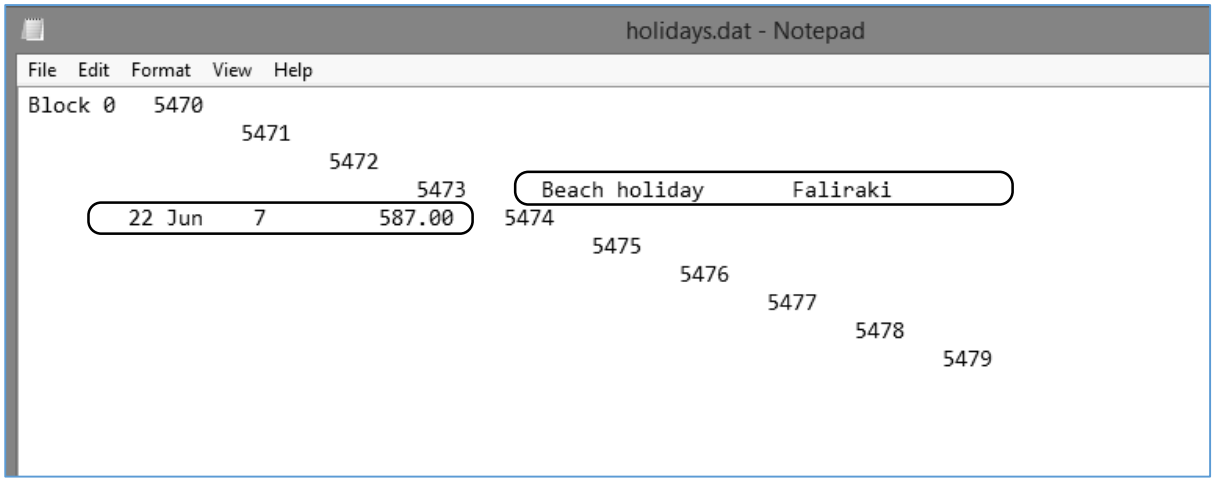
- ***dataBlock*** is the number of the data block where the record is to be stored. For our test record, this will be Block 0.
- ***holidayID***, in our case 5473, will indicate which location within the data block should be used to store the record.
- ***holidayRecord*** is the actual record, containing the holiday category, holiday location, date, legth of holiday and price.

```
private void storeRecord(int dataBlock, int holidayID, String holidayRecord)
{
    try (RandomAccessFile file = new RandomAccessFile(data.holidayFile, "rw"))
    {
        int startPosition=dataBlock*1210;
        int location = holidayID %10;
        int fileposition=startPosition + location*120 + 10;
        file.seek(fileposition);
        file.write(holidayRecord.getBytes());
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(addRecord.this, "File error");
    }
}
```

Run the program. Select the '***Add record***' menu option. Enter a ***HolidayID*** value of ***5473***, then complete the full set of data fields for the holiday record. Click the 'Enter' button. A 'Record saved' message should appear.




Use Windows Explorer to locate the ***holidays.dat*** file in the ***indexedFile*** project folder. Open the file using a text editing application. This file should contain a data block labelled Block 0, with space for inserting holiday records numbered 5470 to 5479. The holiday details which you entered should be present at location 5473 in the file.




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

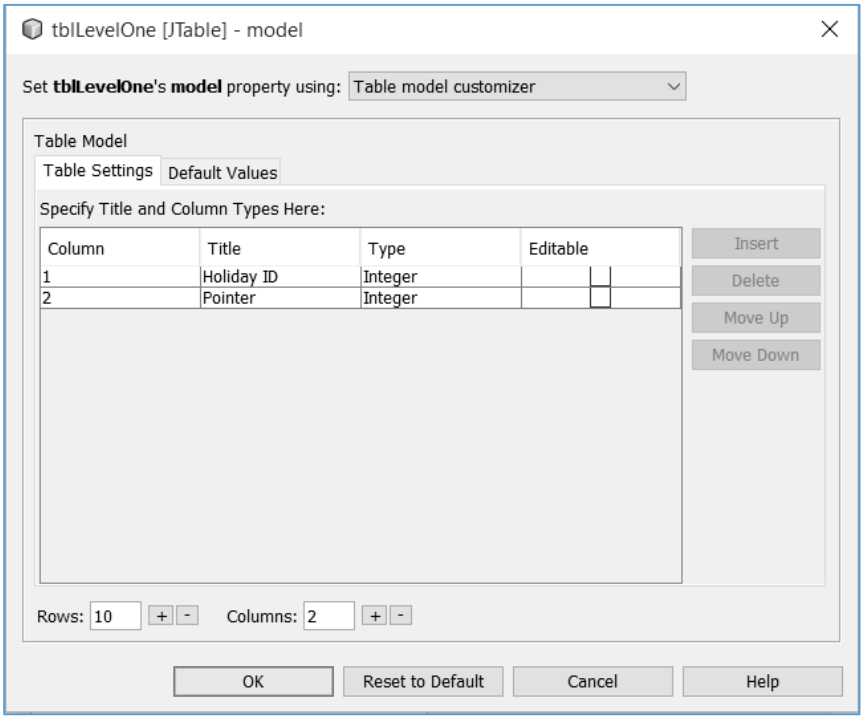
If all has gone well, the program should be storing the holiday records and index values correctly, but it is not very easy to examine the data. We will now add tables to the program screen display, to show the structure of the multi-indexed system more clearly.

Use the tab above the editing screen to move to the ***indexedFile.java*** page, then click the ***Design*** tab to view the form layout screen. Add a ***Table*** component to the screen, renaming this as ***tblLevelOne***. Locate the ***model*** property for the table, and click to open the editing window.

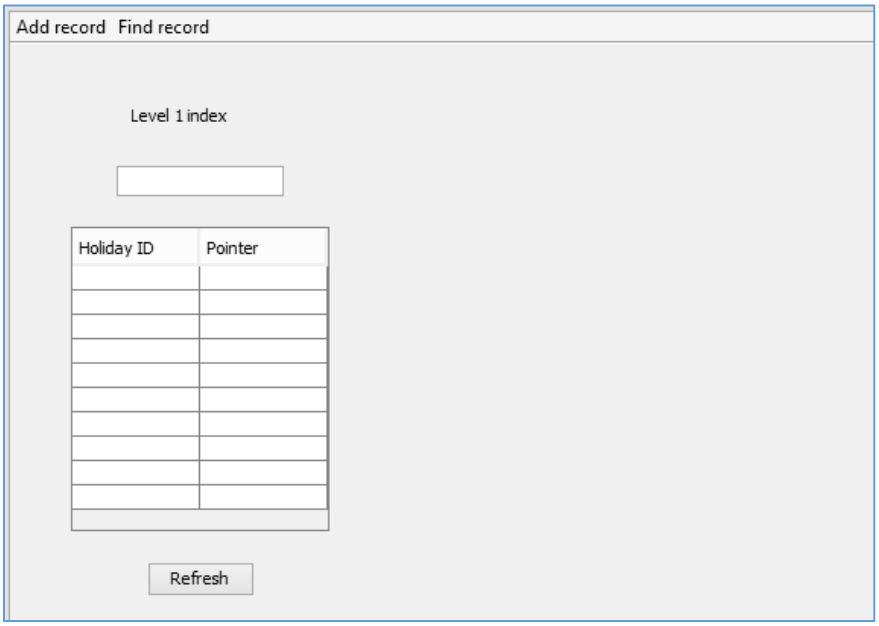
Set the number of ***Rows*** to ***10***, and ***Columns*** to ***2***. Enter the column headings and data types:

| | |
|---|---|
| ***HolidayID*** | ***Integer*** |
| ***Pointer*** | ***Integer*** |

Remove the '***Editable***' ticks from both entries and then click the ***OK*** button.




Add a label '***Level 1 index***' above the table, along with a text field. Rename the text field as ***txtLevelOne***. Add a button below the table with the caption '***Refresh***'. Rename the button as ***btnRefresh***.



Copy the ***label***, ***text field*** and ***table*** components twice more:


The labels should display the captions:

***Level 1 index*** ***Level 2 index*** ***Level 3 index***

The text fields and tables should be named:

***txtLevelOne*** ***txtLevelTwo*** ***txtLevelThree***

***tblLevelOne*** ***tblLevelTwo*** ***tblLevelThree***

Click the ***Source*** tab to move to the program code screen. Go to the top of the program listing and add the Java module which will be needed to operate the List component.

```
package indexedFilePackage;

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

import javax.swing.DefaultListModel;

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

Click the ***Design*** tab to move to the form, then double click the '***Refresh***' button to create a method, then add a line of code to call a ***displayIndexBlock( )*** method. Insert the ***displayIndexBlock( )*** method immediately below the button click method. This method has two parameters: the index block which is to be displayed, and the number of the table where the display will appear.

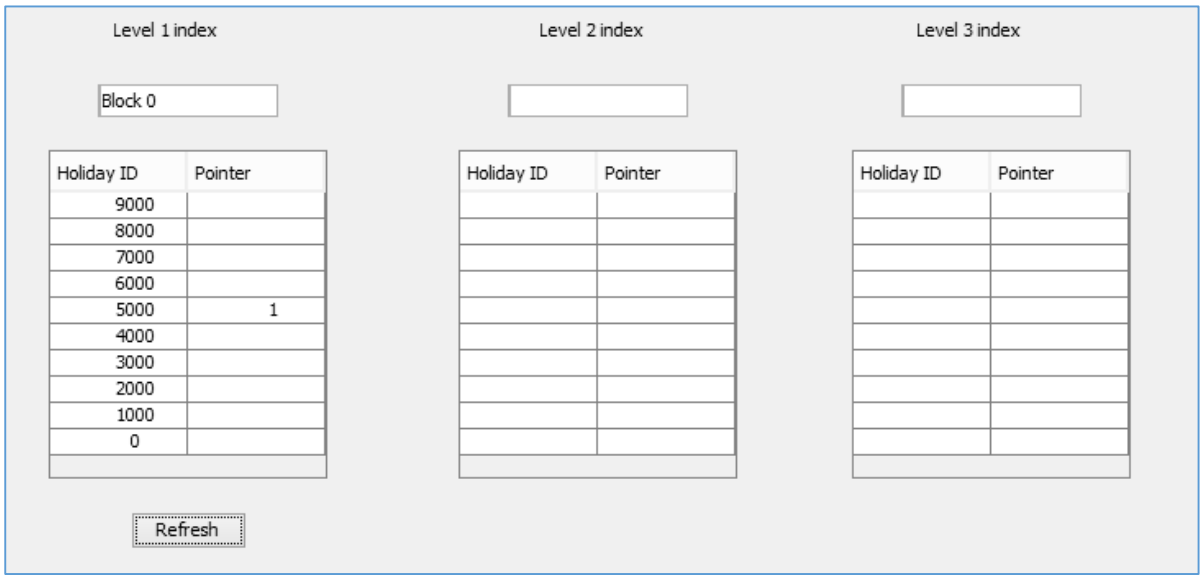
```
private void btnRefreshActionPerformed(java.awt.event.ActionEvent evt) {

    displayIndexBlock(0,1);

}

private void displayIndexBlock(int block, int indexLevel)
{
    try (RandomAccessFile file = new RandomAccessFile(data.indexFile, "r"))
    {
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(indexedFile.this, "File error");
    }
}
```

Add lines of code to the ***displayIndexBlock( )*** method which will:

- Load the required index block from the ***index.dat*** file.
- Display the number of the index block in the text field above the table.
- Carry out a loop 10 times to insert the storage location numbers into the table.
- Display any link pointer values which have been set.

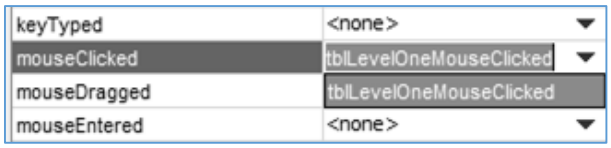
```
private void displayIndexBlock(int block, int indexLevel)
{
    try (RandomAccessFile file = new RandomAccessFile(data.indexFile, "r"))
    {
        byte[] bytes = new byte[210];
        file.seek(block*210);
        file.read(bytes);
        file.close();
        String index=new String(bytes);
        String blockNumber =index.substring(0,10);  index=index.substring(10);
        switch (indexLevel)
        {
            case 1:  txtLevelOne.setText(blockNumber); break;
            case 2:  txtLevelTwo.setText(blockNumber); break;
            case 3:  txtLevelThree.setText(blockNumber); break;
        }

        for (int i=0; i<10; i++)
        {
          String locationNumber=index.substring(0,10);  index=index.substring(10);
          String link=index.substring(0,10);  index=index.substring(10);
          switch (indexLevel)
          {
              case 1:  tblLevelOne.setValueAt(locationNumber,9-i,0); break;
              case 2:  tblLevelTwo.setValueAt(locationNumber,9-i,0); break;
              case 3:  tblLevelThree.setValueAt(locationNumber,9-i,0); break;
          }

          int n=Integer.parseInt(link.trim());
          if (n>=0)
          {
              switch (indexLevel)
              {
                  case 1:  tblLevelOne.setValueAt(link,9-i,1); break;
                  case 2:  tblLevelTwo.setValueAt(link,9-i,1); break;
                  case 3:  tblLevelThree.setValueAt(link,9-i,1); break;
              }
          }
        }
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(indexedFile.this, "File error");
    }
}
```

Run the program, then click the '***Refresh***' button. The Level 1 index should be shown with storage locations numbered at intervals of 1000, from 0 to 9000. Pointer values to Level 2 index blocks will appear for any holiday records which have now been entered.




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

Click the ***Design*** tab to move to the form layout view. We will now set up a method which will display a ***Level 2 index block*** when the corresponding Level 1 table entry is clicked. Select the ***Level 1 table***, then go to the Properties window and click the ***Events*** tab. Locate the ***mouseClicked*** event, and accept ***tblLevelOneMouseClicked*** from the drop down list.




Add lines of code to the ***mouseClicked( )*** method which will:

- Clear any previous entries in the Level 2 and Level 3 tables.
- Find which ***row*** of the Level 1 table has been clicked, and obtain the ***link pointer*** value in the ***right hand column*** of that row.
- Call the ***displayIndexBlock( )*** method to display this Level 2 index block.

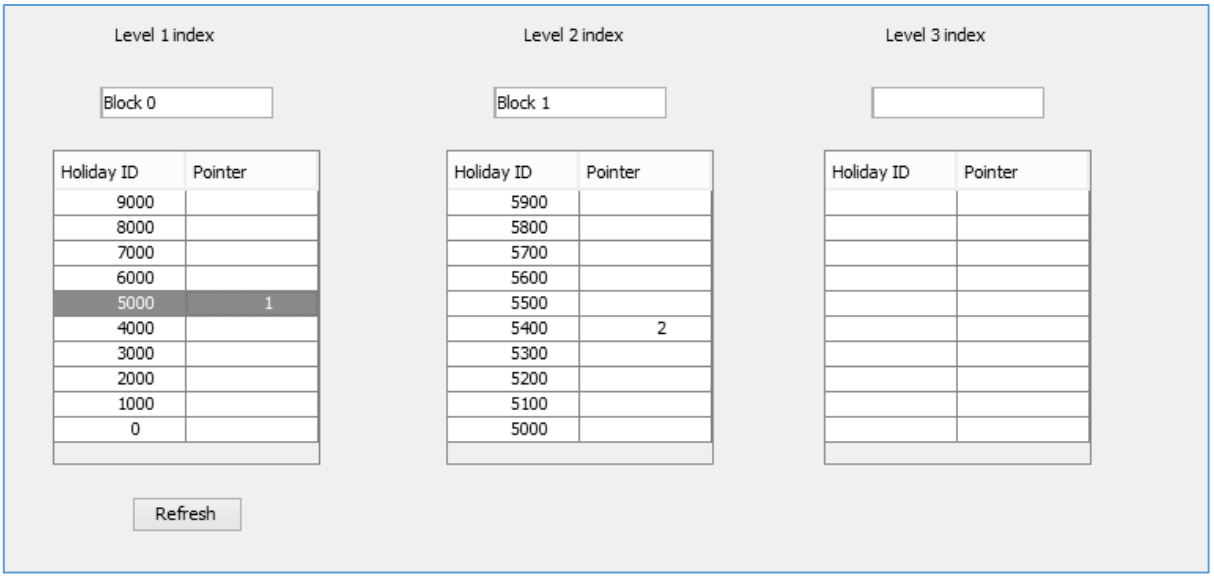
```
private void tblLevelOneMouseClicked(java.awt.event.MouseEvent evt) {
    clearTable2();
    clearTable3();
    try
    {
        int row = tblLevelOne.rowAtPoint(evt.getPoint());
        if (tblLevelOne.getValueAt(row,1)!=null)
        {
            String link=(String) tblLevelOne.getValueAt(row,1);
            int linkBlock=Integer.parseInt(link.trim());
            displayIndexBlock(linkBlock,2);
        }
    }
    catch(NumberFormatException e)
    {
    }
}
```

Add the methods to clear the Level 2 and Level 3 tables immediately below the ***mouseClicked( )*** method.

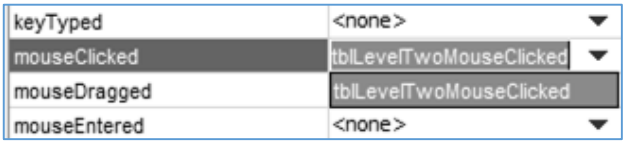
```
private void clearTable2()
{
    for (int i=0;i<10;i++)
    {
         tblLevelTwo.setValueAt("",i,0);
         tblLevelTwo.setValueAt("",i,1);
    }
}

private void clearTable3()
{
    for (int i=0;i<10;i++)
    {
         tblLevelThree.setValueAt("",i,0);
         tblLevelThree.setValueAt("",i,1);
    }
}
```

Run the program, then click the '***Refresh***' button. The Level 1 index should be shown, with a pointer value of 1 for the entry at 5000. Click on this row of the table.

Index block 1 should now be displayed in the Level 2 table, with a link pointer shown for the entry at 5400.




Close the program window and return to the NetBeans editing screen, then click the ***Design*** tab. Select the Level 2 table, then go to the ***Events*** list in the Properties window. Locate the mouseClicked event, and accept tblLevelTwoMouseClicked from the drop down list.



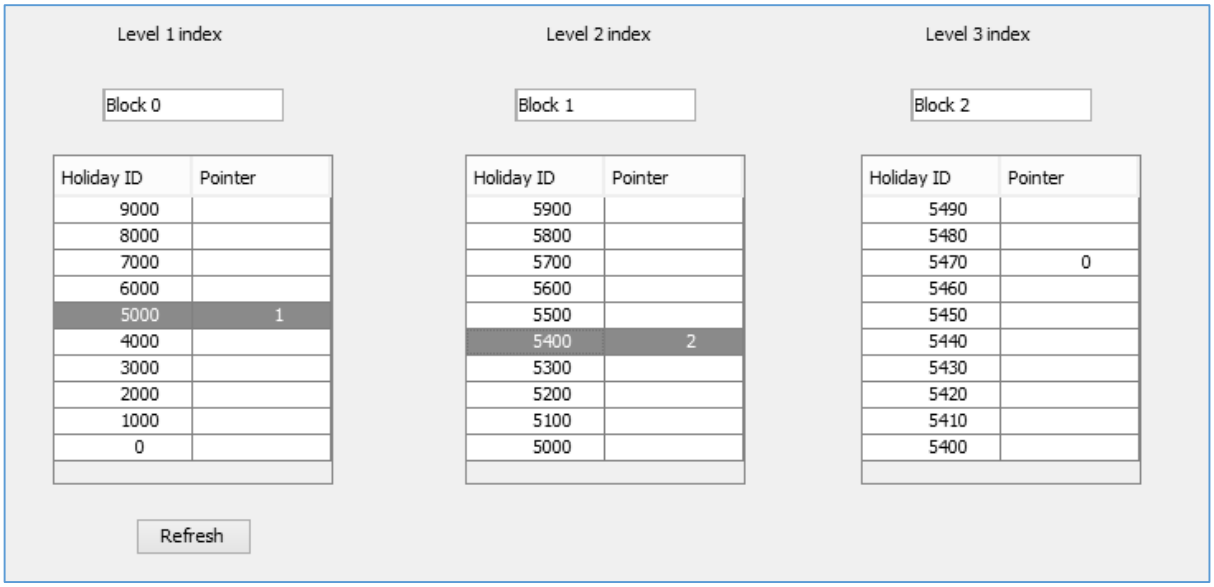
Add lines of code to the ***mouseClicked( )*** method:

```
private void tblLevelTwoMouseClicked(java.awt.event.MouseEvent evt) {
    clearTable3();
    try
    {
        int row = tblLevelTwo.rowAtPoint(evt.getPoint());
        if (tblLevelTwo.getValueAt(row,1)!=null)
        {
            String link=(String) tblLevelTwo.getValueAt(row,1);
            int linkBlock=Integer.parseInt(link.trim());
            displayIndexBlock(linkBlock,3);
        }
    }
    catch(NumberFormatException e)
    {
    }
}
```

Run the program, then click the '***Refresh***' button. The Level 1 index should be shown, with a pointer value for the entry at 5000. Click on this row of the table.

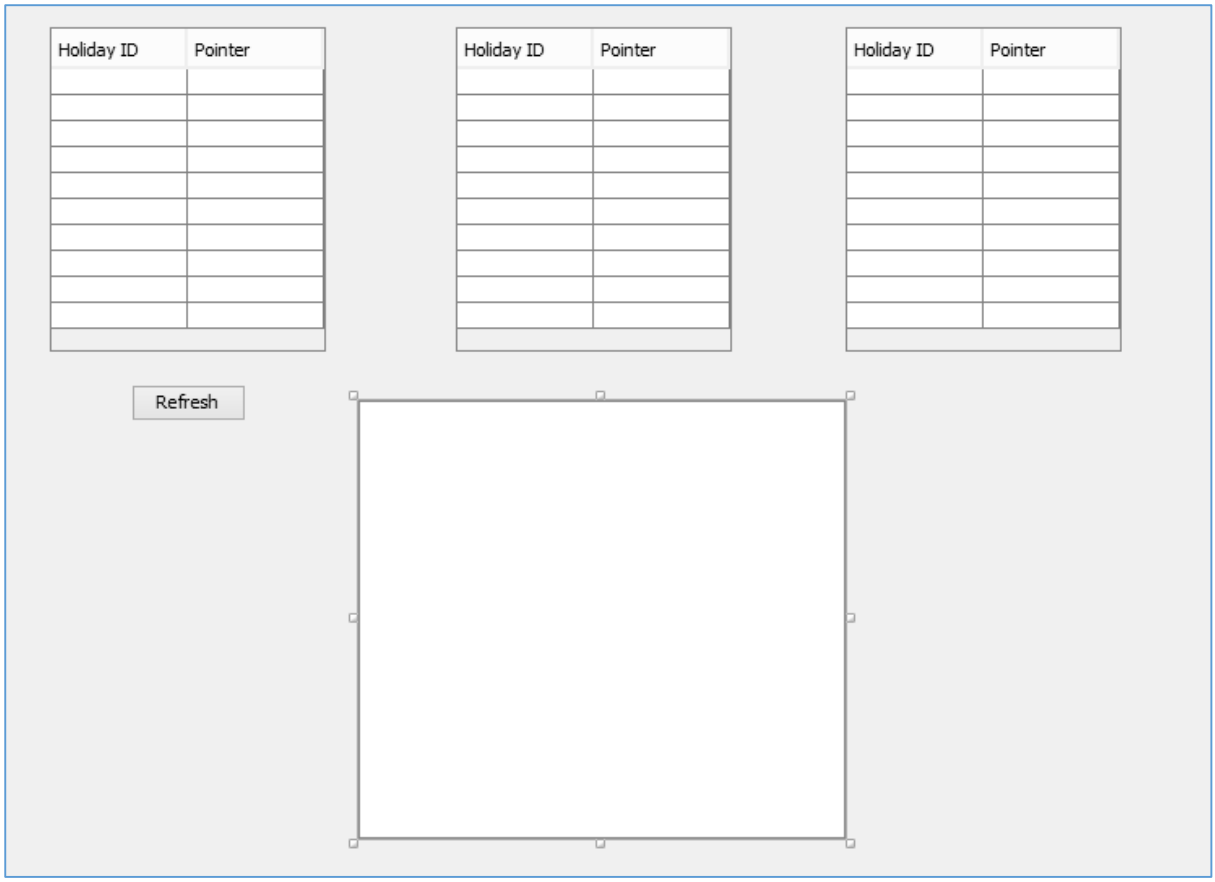
The Level 2 index should now appear, with a pointer value for the entry at 5400. Click on this row of the table.

Entries should now appear in the Level 3 table, with a link pointer to the data block containing records in the range 5470 – 5479.




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

Add a ***List*** component to the form below the three tables. This will be used to display the holiday records found in the data block after a search through the three index levels. Rename the List as ***lstHolidayRecords***.




Select the ***Level 3 table***. Go to the ***Events*** tab and select the ***mouseClicked*** event. Accept ***tblLevelThree MouseClicked*** from the drop down list. Add lines of code which will:

- Clear any previous entries in the list box.
- Find which ***row*** of the Level 3 table has been clicked, and obtain the ***link pointer*** value in the ***right hand column*** of that row.
- Call the ***displayDataBlock( )*** method to display the selected group of 10 holiday records.

```
private void tblLevelThreeMouseClicked(java.awt.event.MouseEvent evt) {
    clearList();
    try
    {
        int row = tblLevelThree.rowAtPoint(evt.getPoint());
        if (tblLevelThree.getValueAt(row,1)!=null)
        {
            String link=(String) tblLevelThree.getValueAt(row,1);
            int linkBlock=Integer.parseInt(link.trim());
            displayDataBlock(linkBlock);
        }
    }
    catch(NumberFormatException e)
    {
    }
}
```

Insert the ***clearList( )*** method below the ***LevelThreeMouseClicked( )*** method.

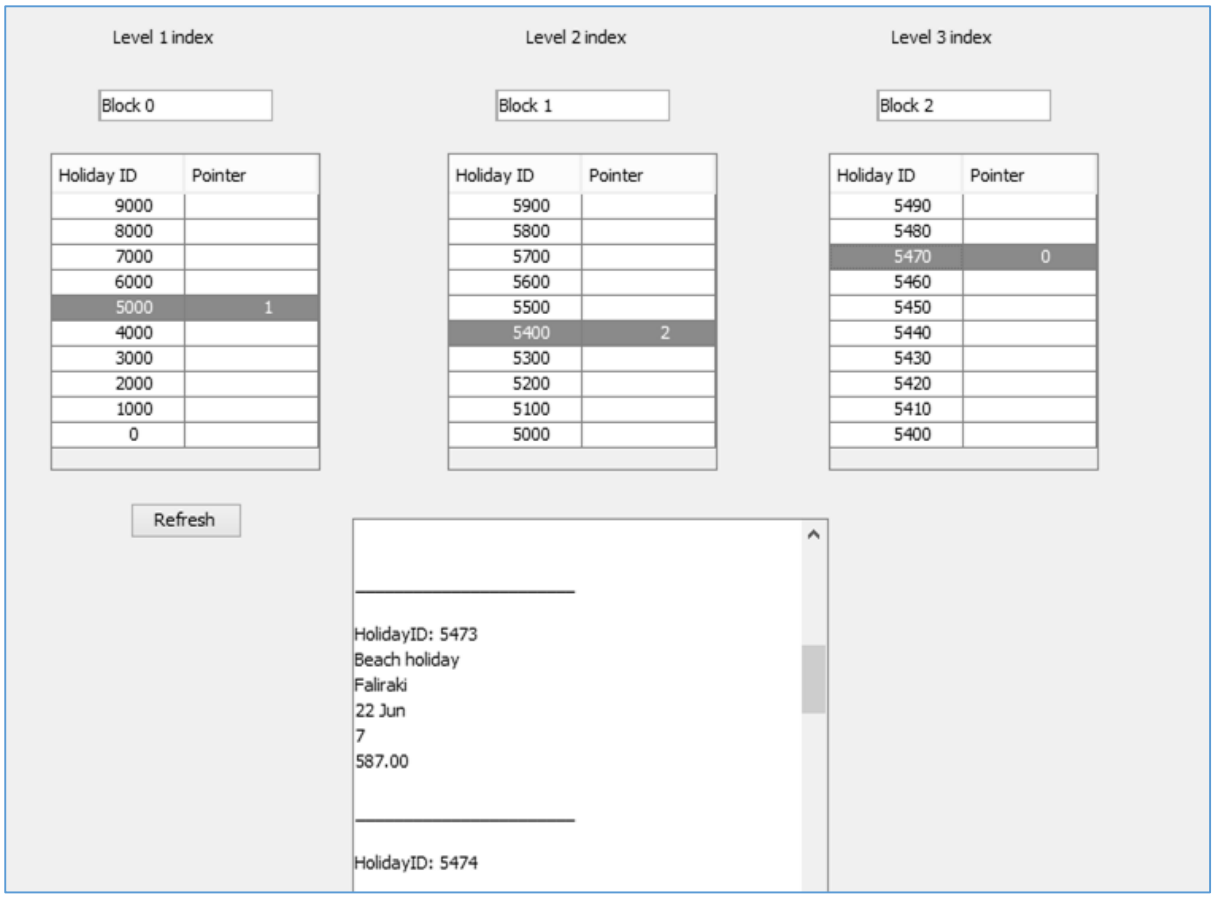
```
private void clearList()
{
    DefaultListModel listModel = new DefaultListModel();
    listModel.clear();
    lstHolidayRecords.setModel(listModel);
}
```

Finally add the ***displayDataBlock( )*** method below the ***clearList( )*** method. This method will:

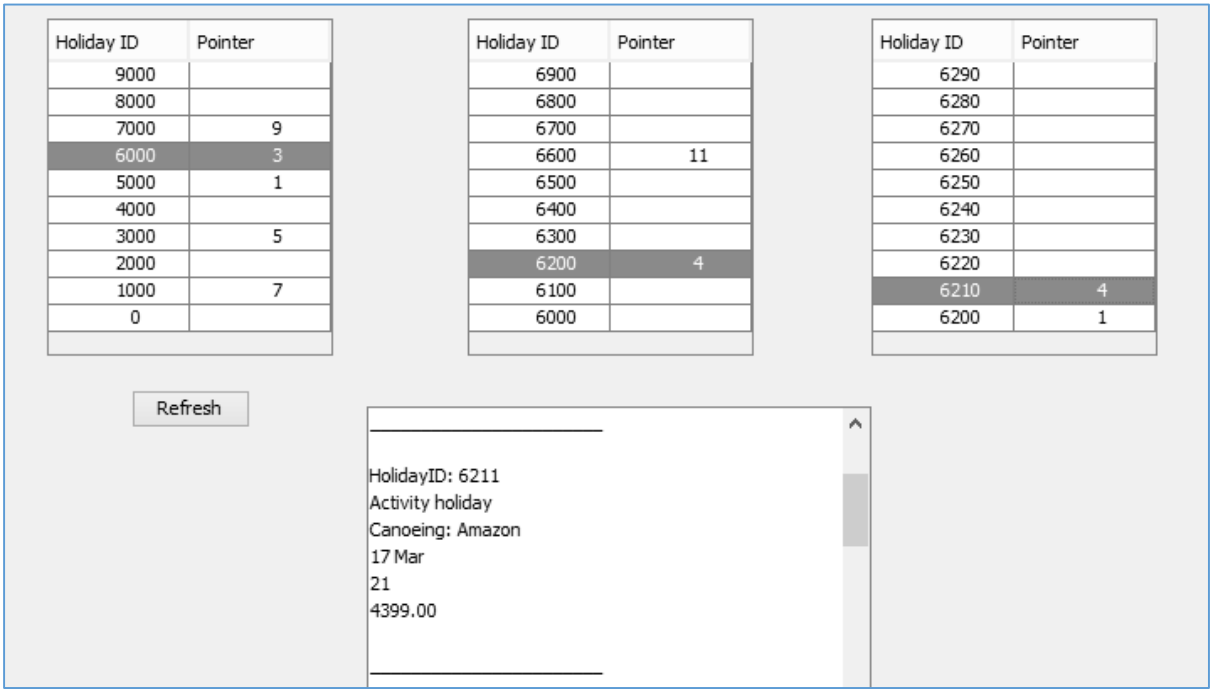
- Load the required data block from the ***holidays.dat*** file.
- Carry out a loop to display each of the 10 records in the data block, splitting each record into its fields: ***holidayID***, holiday ***category*** and ***location***, ***start date*** and ***length*** of holiday, and the ***price***.

```
private void displayDataBlock(int block)
{
    DefaultListModel listModel = new DefaultListModel();
    String s;
    try (RandomAccessFile file = new RandomAccessFile(data.holidayFile, "r"))
    {
        byte[] bytes = new byte[1210];
        file.seek(block*1210);
        file.read(bytes);
        file.close();
        s=new String(bytes);
        file.close();
        String blockNumber =s.substring(0,10);  s=s.substring(10);
        listModel.addElement(blockNumber);
        for (int i=0; i<10; i++)
        {
          String holidayID=s.substring(0,10);  s=s.substring(10);
          listModel.addElement("HolidayID: "+holidayID);
          String category=s.substring(0,20);  s=s.substring(20);
          listModel.addElement(category);
          String location=s.substring(0,60);  s=s.substring(60);
          listModel.addElement(location);
          String startDate=s.substring(0,10);  s=s.substring(10);
          listModel.addElement(startDate);
          String days=s.substring(0,10);  s=s.substring(10);
          listModel.addElement(days);
          String price=s.substring(0,10);  s=s.substring(10);
          listModel.addElement(price);
          listModel.addElement(" ");
          listModel.addElement("_________________________");
          listModel.addElement(" ");
        }
        lstHolidayRecords.setModel(listModel);
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(indexedFile.this, "File error");
    }
}
```

Run the program. Click the '***Refresh***' button, then work through the series of index levels to reach the data block for record ***5473***. Check that the fields of this record are shown correctly in the List.

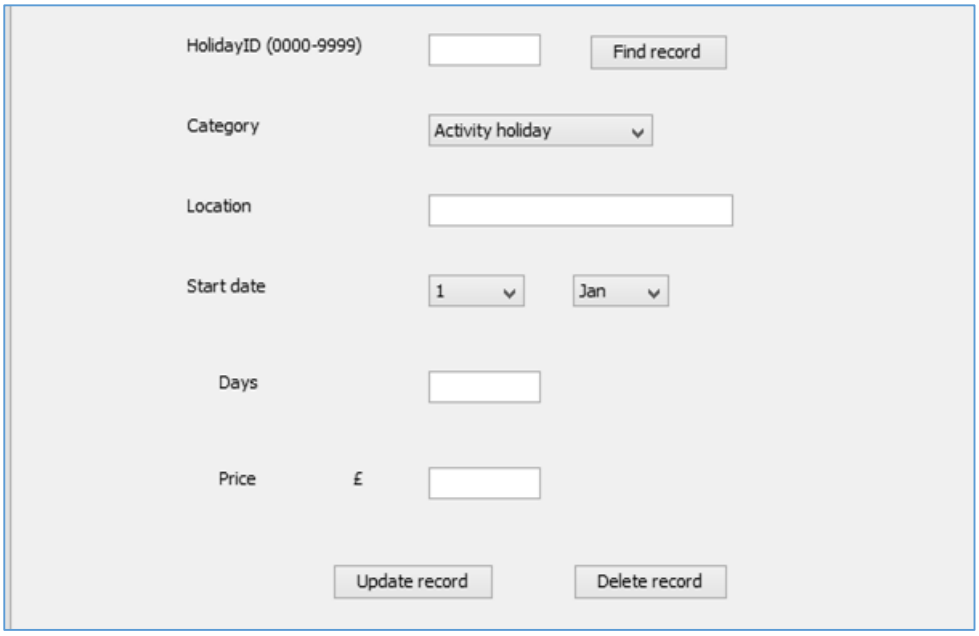



Add further holiday records using the '***Add record***' menu option. Choose ***holidayID*** values spread over the range of possible numbers from 0000 to 9999. After each new record is entered, click the '***Refresh***' button on the main program page and check that this record can be accessed through the menu system.



Close the program windows and return to the NetBeans editing screen. Our last task is to provide a page where records can be displayed, then updated or deleted as required.

Use the tab above the editing screen to move to the ***findRecord.java*** page. Add components to the form as shown:




- A label '***HolidayID (0000 – 9999)***', with a text field alongside. Rename the text field as ***txtHolidayID***. Alongside the text field add a button with the caption '***Find record***'. Rename the button as ***btnFindRecord***.
- A label '***Category***' with a Combo Box alongside. Rename the Combo Box as ***cmbCategory***. Select the Combo Box and click the ellipsis (…) symbol at the end of the ***model*** property row to open an editing window. List the categories of holiday: ***Activity holiday***, ***Beach holiday***, ***Touring***, ***Cruise***.
- A label '***Location***', with a text field alongside. Rename the text field as ***txtLocation***.
- A label '***Start date***'. Place two Combo Boxes alongside, renaming these as ***cmbDay*** and ***cmbMonth***. Select each of the Combo Boxes in turn, clicking the ellipsis (…) symbol at the end of the ***model*** property row to open the editing window. Enter the numbers ***1*** to ***31*** as the drop down list options for ***cmbDay***, and the month names ***Jan*** to ***Dec*** for ***cmbMonth***.
- A label '***Days***', with a text field alongside. Rename the text field as ***txtDays***.
- A label '***Price***', with a text field alongside. Rename the text field as ***txtPrice***.
- Buttons with the captions '***Update record***' and '***Delete record***'. Rename the buttons as ***btnUpdate*** and ***btnDelete***.

Use the Source tab to move to the program code page. We will add Java modules at the beginning of the program listing which will be needed for file handling, and to display message boxes. Also add variables which will be needed by the program, as shown below.

```
package indexedFilePackage;

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

public class findRecord extends javax.swing.JFrame {

    int holidayID = -1;
    int dataBlock;

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

Use the Design tab to return to the form layout view. Double click the '***Find record***' button to create a method, then add a line of code to call a ***findRecord( )*** method.

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

        findRecord();

    }
```

Add the ***findRecord( )*** method immediately below the buttonClick method. This method will:

- obtain the ***holidayID*** from the input text field,
- check that the ***holidayID*** number has a correct format,
- call a ***searchIndex( )*** method to find the location of the data block containing the required record, then
- load the data block and display the required record.

```
private void findRecord()
{
    String holidayCode=txtHolidayID.getText();
    holidayCode.trim();
    if (holidayCode.length()!=4)
    {
        JOptionPane.showMessageDialog(findRecord.this,
                                      "The holiday code must be four digits");
    }
    else
    {
        try
        {
            holidayID = Integer.parseInt(holidayCode);
            dataBlock=searchIndex(holidayID);
            displayRecord(holidayID, dataBlock);
        }
          catch(NumberFormatException e)
        {
            JOptionPane.showMessageDialog(findRecord.this,
                                                  "Incorrect number format");
        }
    }
}
```

Please note that the lines beginning '***JOptionPane.showMessageDialog(…***' in the ***findRecord( )*** method should be entered as single lines of code with no line breaks.

Insert the ***searchIndex( )*** method below the ***findRecord( )*** method. This uses a similar search method to the main program page, working through each of the index levels in turn, and then returning the number of the data block where the required record is stored.

```
private int searchIndex(int holidayID)
{
    int pointer=-1;
    int level1=(holidayID /1000)*1000;
    int indexBlock=0;
    pointer=getPointer(indexBlock,level1);
    if (pointer>0)
    {
        int level2=(holidayID /100)*100;
        indexBlock=pointer;
        pointer=getPointer(indexBlock,level2);
        if (pointer>0)
        {
            int level3=(holidayID /10)*10;
            indexBlock=pointer;
            pointer=getPointer(indexBlock,level3);
        }
    }
    return pointer;
}
```

The ***searchIndex( )*** method in turn requires a ***getPointer( )*** method. We will insert this below the ***searchIndex( )*** method. ***GetPointer( )*** has two parameters:

- ***indexBlock***, which is the number of the index block being searched
- ***searchValue***, which is the location number required.

Begin the method by adding lines of code to open the index file and collect the required index block.

```
private int getPointer(int indexBlock, int searchValue)
{
    int pointer=0;
    byte[] bytes=null;
    try (RandomAccessFile file = new RandomAccessFile(data.indexFile, "r"))
    {
        bytes = new byte[210];
        file.seek(indexBlock*210);
        file.read(bytes);
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(findRecord.this, "File error");
    }
    return pointer;
}
```

We will now add code to the ***getPointer( )*** method to check each of the entries in the index block until the required location value is found. The method then returns the link pointer corresponding to this location.

```
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(findRecord.this, "File error");
    }
    String s = new String(bytes);
    int location;
    String w,x;
    s=s.substring(10);
    boolean found=false;
    while(found==false)
    {
         w=s.substring(0,10);s=s.substring(10);
         x=s.substring(0,10);s=s.substring(10);
         location=Integer.parseInt(w.trim());
         pointer=Integer.parseInt(x.trim());
         if (location==searchValue)
         {
            found=true;
         }
    }
    return pointer;
}
```

We now know the data block number containing the required holiday record. The final step is to load and display the record.

Begin a ***displayRecord( )*** method immediately below the ***getPointer( )*** method. This has two parameters:

- ***holidayID***, which is the ID number of the required holiday record.
- ***dataBlock***, which is the number of the data block where this record is stored.

Add lines of code which will load the required record from the ***holidays.dat*** file.

```
private void displayRecord(int holidayID, int dataBlock)
{
    String s;
    try (RandomAccessFile file = new RandomAccessFile(data.holidayFile, "r"))
    {
        byte[] bytes = new byte[110];
        int position=holidayID % 10;
        file.seek(dataBlock*1210+ position*120 + 20);
        file.read(bytes);
        file.close();
        s=new String(bytes);
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(findRecord.this, "File error");
    }
}
```

The essential lines of code to carry out this operation are:

***byte[] bytes = new byte[110]***

The fields of the record which are being loaded have a total size of 110 bytes:

| Category | Location | Start Date | Days | Price |
|---|---|---|---|---|
| 20 bytes | 60 bytes | 10 bytes | 10 bytes | 10 bytes |

***int position=holidayID % 10***

We find the sequence number of the required record within the data block by calculating the remainder when the ***holidayID*** is divided by 10. The ***%*** sign is the Java ***MOD*** operator, used to calculate a remainder during division.

For example:

***5473 MOD 10*** = ***3***, so the holiday with ID number ***5473*** will be at ***position 3*** within the data block.

***file.seek(dataBlock*1210+ position*120 + 20)***

The start position of the required record, measured in bytes from the start of the holidays.dat file, is calculated. The result is made up from:

***1210 bytes*** missed for each data block before the required block.

Once the start of the required block is reached,

***10 bytes*** missed because of the block heading, e.g. 'Block 0', which occurs before the records start.

***120 bytes*** missed for each record which occur before the required record.

***10 bytes missed*** because we do not need to load the HolidayID value. This is already known.

***file.read(bytes)***

Loads 110 bytes for the required fields of the record.

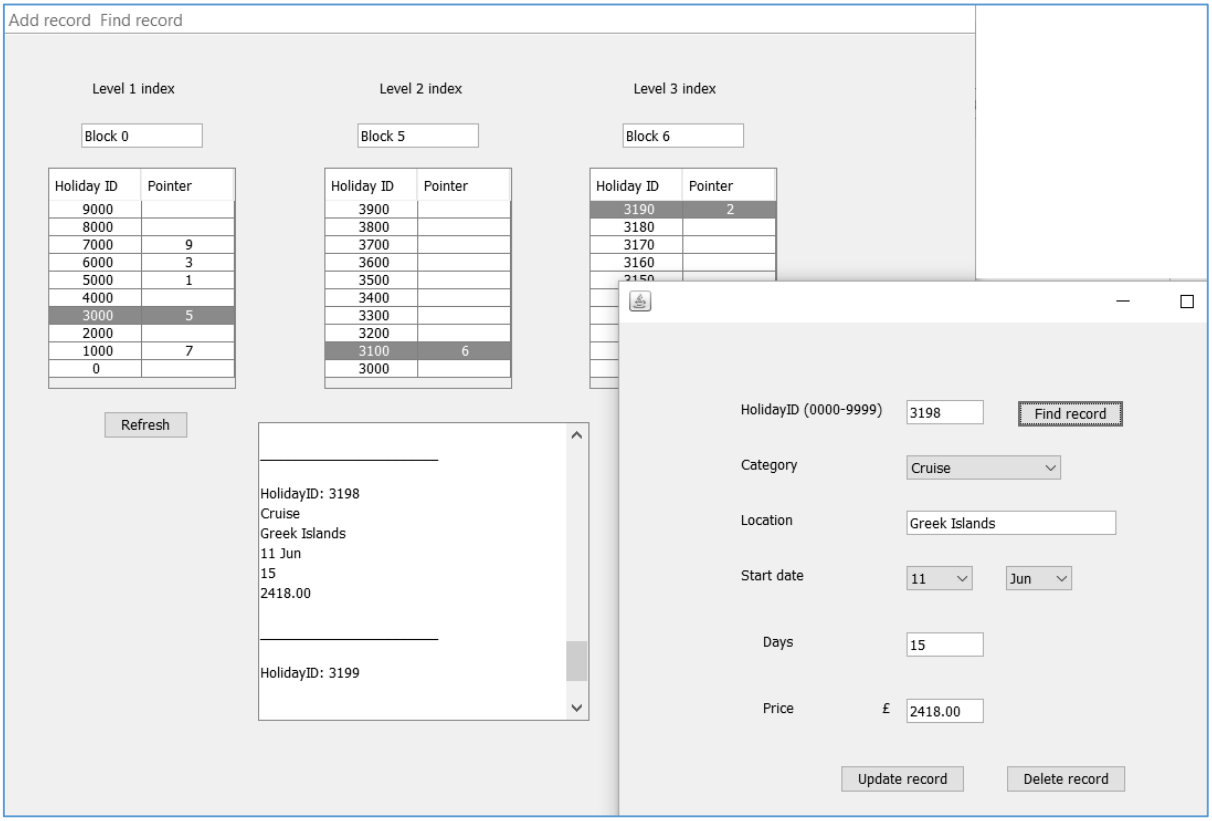
We can now insert lines of code to display the fields of the record in the combo boxes and edit boxes on the form.

```
            file.close();
            s=new String(bytes);
            file.close();
            String category=s.substring(0,20);  s=s.substring(20);
            cmbCategory.setSelectedItem(category.trim());
            String location=s.substring(0,60);  s=s.substring(60);
            txtLocation.setText(location.trim());
            String startDate=s.substring(0,10);  s=s.substring(10);
            String day=startDate.substring(0,3); startDate=startDate.substring(3);
            String month=startDate.trim();
            day=day.trim();
            cmbDay.setSelectedItem(day);
            cmbMonth.setSelectedItem(month);
            String days=s.substring(0,10);  s=s.substring(10);
            txtDays.setText(days);
            String price=s.substring(0,10);
            txtPrice.setText(price);
        }
        catch(IOException e)
        {
              JOptionPane.showMessageDialog(findRecord.this, "File error");
        }
    }
```

Run the program. Click the '***Refresh***' button on the min page and use the tables to locate a holiday record.

Select the '***Find record***' menu option. Enter the corresponding ***HolidayID*** number in the text field, then click the '***Find record***' button. Check that the holiday details are displayed correctly on the form.

Test the loading of each of the other holiday records which you have entered.




Close the program windows and return to the NetBeans editing screen. We will now produce the method to update records.

Use the ***Design*** tab to move to the form layout view. Double click the '***Update record***' button to create a method. Add a line of code to call an ***update( )*** method.

```
private void btnUpdateActionPerformed(java.awt.event.ActionEvent evt) {
    update();
}
```

We will insert the ***update( )*** method immediately below the button click method, as shown below. The method collects the data from the combo boxes and text fields on the form, assembles this into a complete record, then calls a ***storeRecord( )*** method to transfer the updated record back into the data block.

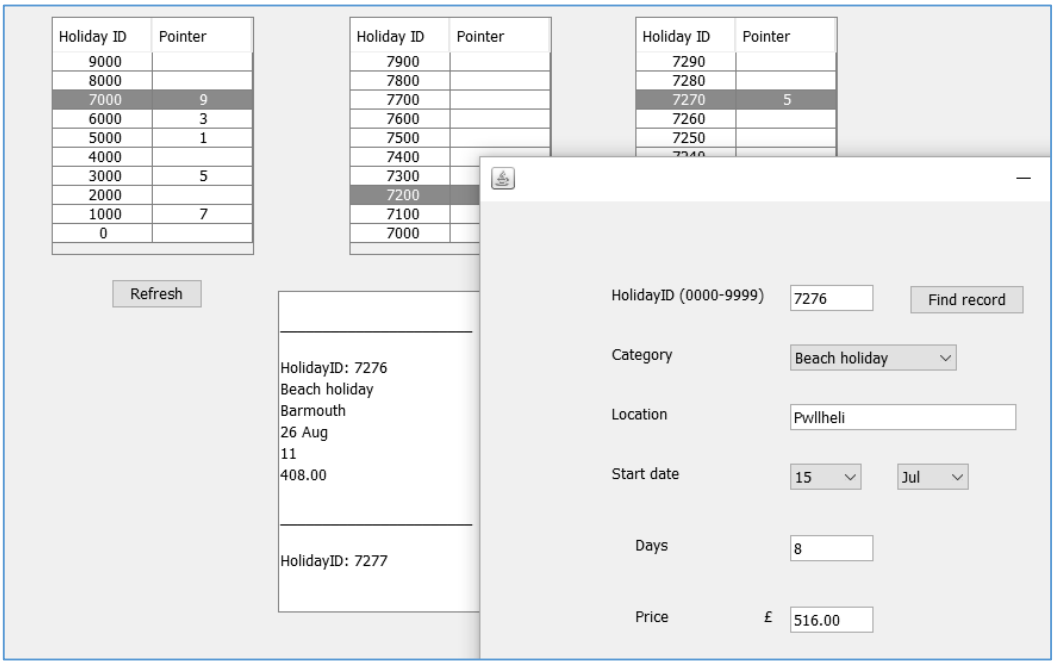
Notice that we are not allowing the ***holidayID*** field to be changed. If the user has made an error in entering this field, the record must be deleted and re-entered with a correct ***holidayID*** value.

```
private void update()
{
   if (holidayID>=0)
   {
           String category=cmbCategory.getSelectedItem().toString();
           String location=txtLocation.getText();
           String startDate=cmbDay.getSelectedItem()+" "+cmbMonth.getSelectedItem();
           String days=txtDays.getText();
           String price=txtPrice.getText();
           category=String.format("%-20s", category);
           location=String.format("%-60s", location);
           startDate=String.format("%-10s", startDate);
           days=String.format("%-10s", days);
           price=String.format("%-10s", price);
           String holidayRecord= category+location+startDate+days+price;
           storeRecord(dataBlock, holidayID, holidayRecord);
           JOptionPane.showMessageDialog(findRecord.this,"Record updated");
           txtLocation.setText("");
           txtDays.setText("");
           txtPrice.setText("");
   }
   else
   {
        JOptionPane.showMessageDialog(findRecord.this,"Record not found");
   }
}
```

Add the ***storeRecord( )*** method immediately after the ***update( )*** method. This method uses the same calculation which was discussed earlier for the ***displayRecord( )*** method, to determine the position in the holidays.dat file where the updated record should be stored.

```
private void storeRecord(int dataBlock, int holidayID, String holidayRecord)
{
    try (RandomAccessFile file = new RandomAccessFile(data.holidayFile, "rw"))
    {
        int startPosition=dataBlock*1210;
        int location = holidayID %10;
        int fileposition=startPosition + location*120 + 20;
        file.seek(fileposition);
        file.write(holidayRecord.getBytes());
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(findRecord.this, "File error");
    }
}
```

Run the program and use the index tables to locate a holiday record which has been saved. Go to the '***Find record***' option, load the record and make changes to several of the fields. Click the Update button, then close the program windows. Re-run the program and check that the record has been updated correctly.




Close the program windows and return to the NetBeans editing screen. It just remains to add a delete method. Use the ***Design*** tab to move to the form layout view, then double click the '***Delete record***' button to create a method. Add a line of code to call a ***delete( )*** method, then insert the delete( ) method immediately underneath.

```
private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
   delete();
}
```

```
private void delete()
{
     if (holidayID>=0)
     {
         int response = JOptionPane.showConfirmDialog(null,
                   "Are you sure you want to delete this record?", "Confirm",
                   JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
         if (response == JOptionPane.YES_OPTION)
         {
             String holidayRecord =String.format("%-110s", "");
             storeRecord(dataBlock, holidayID, holidayRecord);
             JOptionPane.showMessageDialog(findRecord.this,"Record deleted");
             txtLocation.setText("");
             txtDays.setText("");
             txtPrice.setText("");
         }
     }
     else
     {
          JOptionPane.showMessageDialog(findRecord.this,"Record not found");
     }
}
```

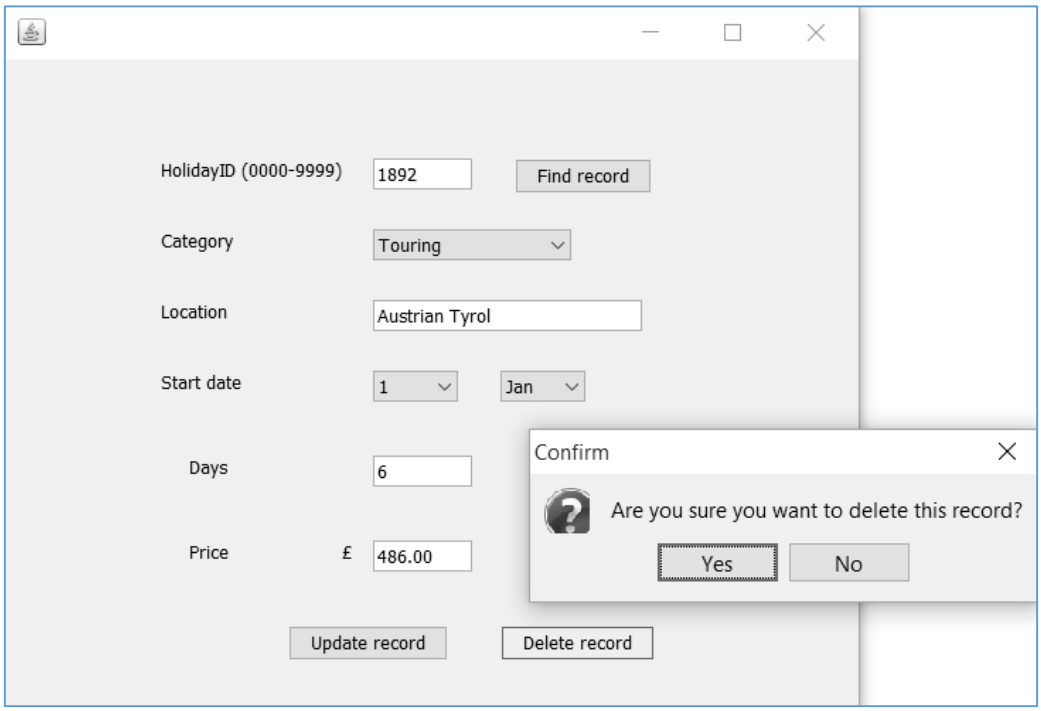
Please note that the line beginning:

***int response = JOptionPane.showConfirmDialog(…***

should be entered as a single line of code without line breaks.

The ***delete( )*** method works in a very similar way to the ***update( )*** method, except that a blank record made up of 110 space characters is inserted in place of an actual record.

Run the program. Select a record using the index tables, then go to the '***Find record***' page. Load the record and click the '***Delete***' button. Confirm to delete the record.




Close the program windows, then re-run the program. Check that the record has been deleted from the corresponding data block location.

In this project we have demonstrated that a selected record can be quickly loaded from a data store containing up to 10,000 records. This system could readily be scaled up to handle systems involving hundreds of thousands, or even millions, of records.