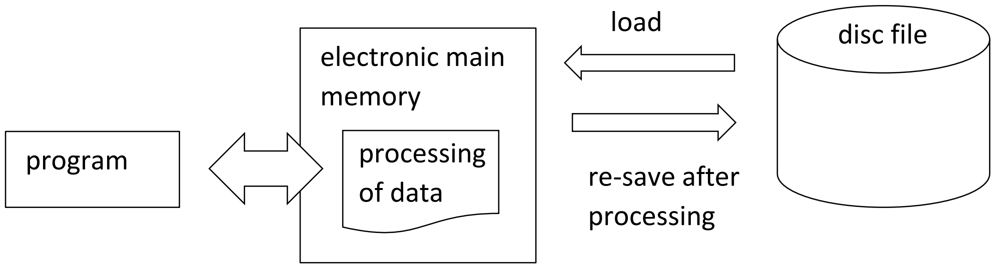
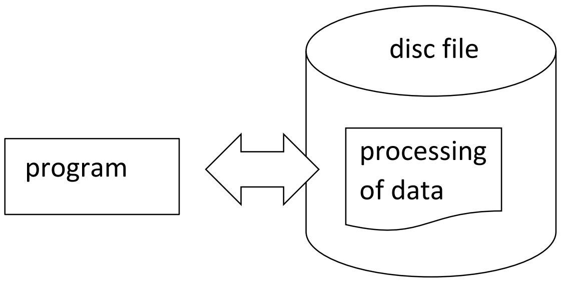
16 Random access files

When a program runs, a set of data can be loaded from disc into the ***electronic main memory*** of the computer. Here it can be ***sorted***, ***searched*** or ***updated***, and any changes saved back to the disc file.



In previous projects we have been largely working with data in electronic memory, using either ***arrays*** or ***objects*** as temporary storage for our data. This approach has considerable advantages, as the processing of a set of records can be thousands of times faster in the electronic main memory than if data has to be constantly accessed from a disc file using slow read and write operations.

Sometimes, however, the amount of data being handled is too large to all be held in the main memory at once. A system may handle many thousands or even millions of records. Examples of such large systems might include Government Social Security records, Driver and Vehicle Licensing records, customers' policy records for a large Insurance Company, or product records for a major Supermarket chain. In these situations, the program must directly access individual records ***from the disc file*** during processing.

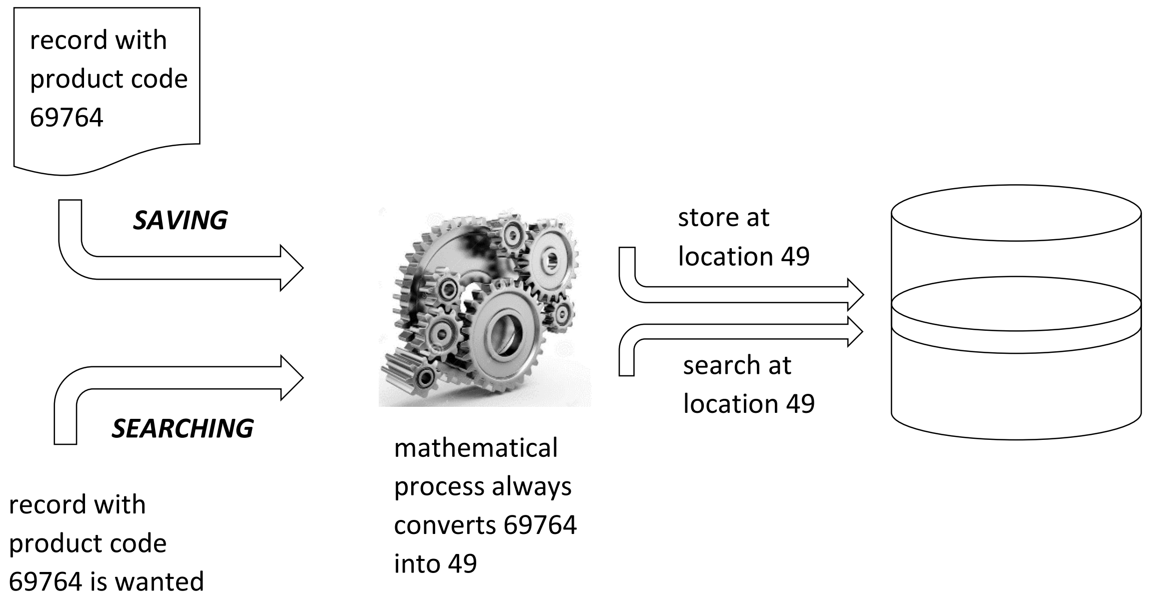


A major problem for this type of system is that disc operations are slow, due to the limited speed with which the mechanical components of the disc drive can move to access particular records. For a disc based system to work efficiently, the number of disc operations must be kept to an absolute minimum. When reading a record from disc, we need to know its likely location on the disc so that it can be loaded in a single operation. Access will be slow if it is necessary to check many different locations on the disc before finding the required record. In this and the next chapter we will look at two systems which have been developed to provide fast access to particular records on disc: ***random access files*** and ***indexed sequential files***.

Random access files

The clever idea of random access files is to take a ***value from each record*** and convert this into a ***file location*** by some ***mathematical process***. For example:

> Suppose that a particular stock record contains a ***product code*** with the value ***69764***. A ***mathematical process*** could be devised which always ***converts this number*** into some other small value such as ***49***. The record would be ***stored*** in the calculated file location, 49. When the record needs to be ***accessed***, the required product code is input to the same mathematical process, and the number 49 is again calculated. It should then be possible to go directly to location 49 to obtain the record.


record with product code 69764
SAVING
SEARCHING
record with product code 69764 is wanted
mathematical process always converts 69764 into 49
store at location 49
search at location 49

The mathematical process used by a random access file system is called a ***hash function***. Many different mathematical processes could be used, but a convenient method is to find the ***remainder after division***. This can be calculated easily by means of the Java ***MOD*** function.

> Suppose that a shop expects to stock a maximum of ***1000 different products***. A random access file system could be set up using ***1000 memory locations***, numbered 0 – 999.
>
> Stock items may have six digit product codes, such as 389231.
>
> A suitable ***hash function*** for calculating the storage location for any product would be:
>
> ***<product code> MOD 1000***
>
> The remainder after dividing 389231 by 1000 will be 231, so the product with stock code ***389231*** is stored at location ***231***.

You may have spotted a potential problem with this system. Other stock codes, for example ***461231*** or ***788231***, would also generate the same value of ***231*** when the hash function is applied. If two records generate the same hash value, it is said that a ***collision*** has occurred. Collisions are quite rare if the random access file is created with at least ***one third more memory locations*** than the expected maximum number of records to be stored. However, collisions can still occur, and a strategy is needed for dealing with these.

In this chapter we will set up a ***random access file*** system to handle product records for an on-line book, music and electronic entertainment store. The store stocks approximately ***600 different items***, and each has been given a ***six digit stockID*** number. We will explore how records can be saved into a disc file, accessed from disc, and collisions handled by the system.

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

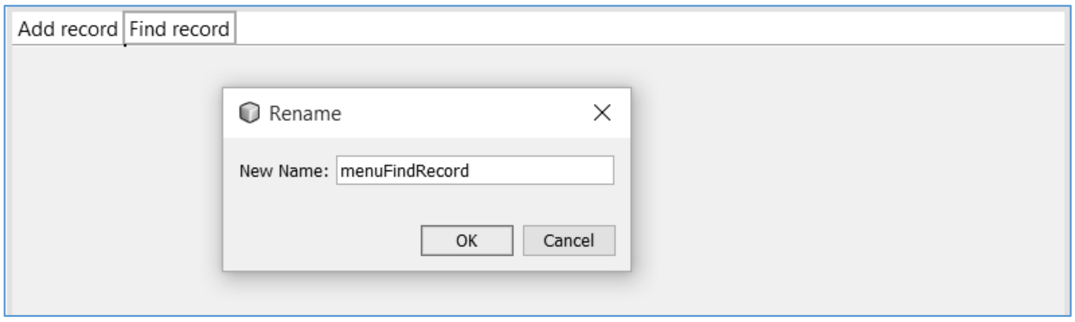
Return to the NetBeans editing page.   Right-click on the ***randomAccess*** project, and select ***New / JFrame Form***. Give the ***Class Name*** as ***randomAccess***, and the ***Package*** as ***randomAccessPackage***:

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***".

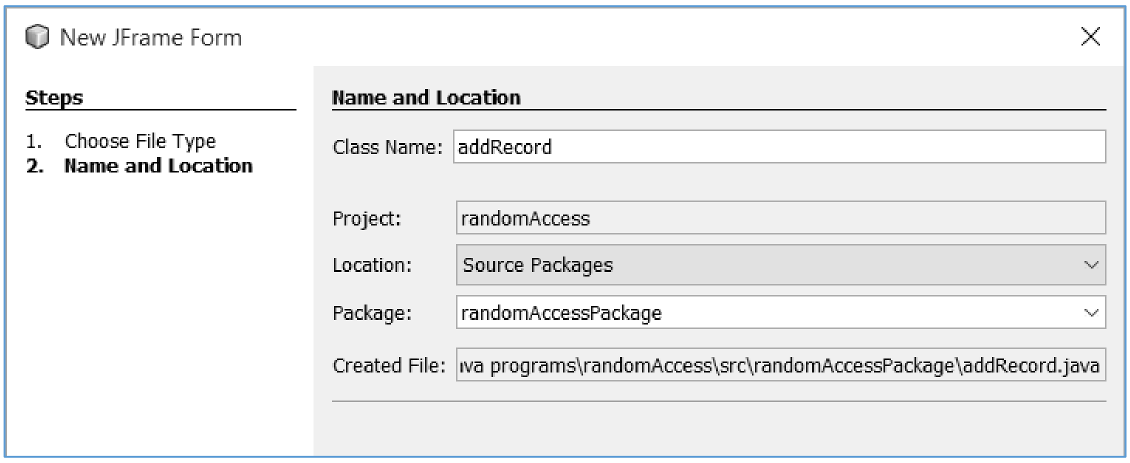
Run the program and accept the ***main*** class which is offered. Check that a blank window appears and has the correct size and colour scheme. Close the program and return to the editing screen.

Click the Design tab to move to the form layout view. Add a Menu Bar component. Right-click on the menu items and change the text entries to '***Add record***' and '***Find record***'. Rename the menu items as ***menuAddRecord*** and ***menuFindRecord***.



New Name: menuFindRecord

Go to the Projects window at the top left of the editing screen and right-click on the randomAccessPackage folder. Select ***New / JFrame Form***. Give the Class Name as ***addRecord***, and leave the Package name as ***randomAccessPackage***.



Class Name: addRecord

Project: randomAccess

Location: Source Packages

Package: randomAccessPackage

Created File: ıva programs\randomAccess\src\randomAccessPackage\addRecord.java

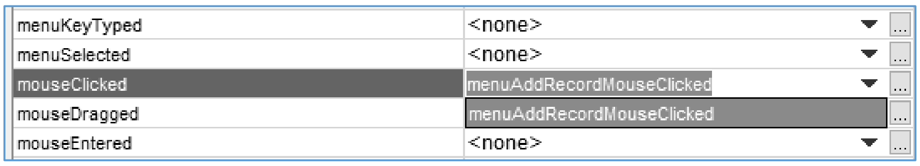
Click the Finish button to return to the editing screen. The new ***addRecord*** form should appear.

- 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***'.

Return to the Projects window and again right-click on the ***randomAccessPackage*** folder. Select ***New*** / ***JFrame Form*** to create another blank form. Give the Class Name as ***findRecord***, and leave the Package name as ***randomAccessPackage***. Click the ***Finish*** button to create the form.

Choose the options ***Set layout*** / ***Absolute layout***, ***Form Size Policy*** / ***Generate pack()*** / ***Generate Resize code*** and ***defaultCloseOperation***/ '***HIDE***'.

Click the tab at the top of the editing screen to open the ***randomAccess.java*** form. Select the '***Add record***' menu option. Go to the Properties window and click on the ***Events*** tab. Locate the ***mouseClicked*** event and accept ***menuAddRecordMouseClicked*** from the drop down list.



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

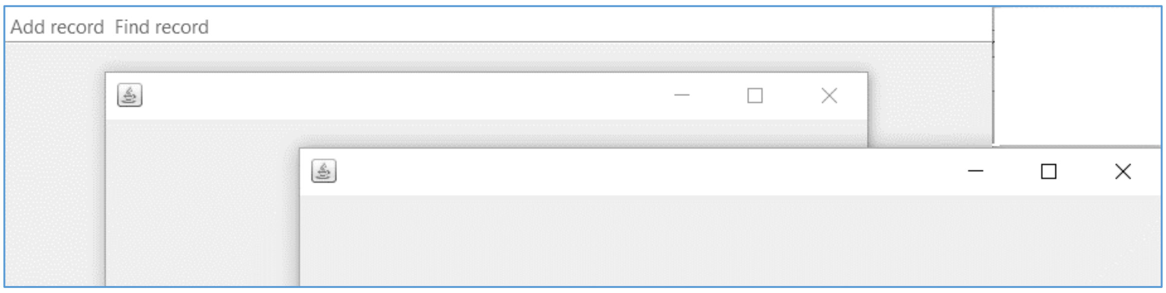
Add a line of code to the ***mouseClicked*** method to open the ***addRecord*** form.

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

Click the Design tab to return to the form layout view, then repeat the procedure to produce a ***mouseClicked*** method for the '***Find record***' menu item. Add a line of code to open the ***findRecord*** form.

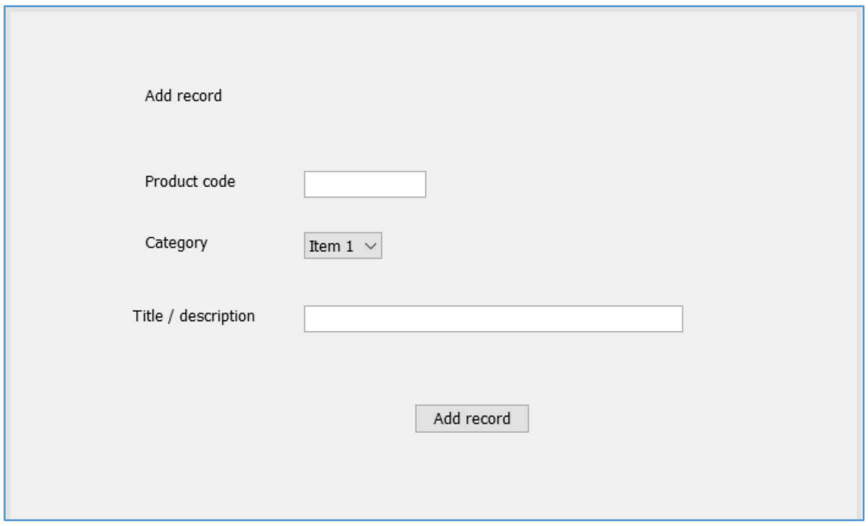
```
private void menuFindRecordMouseClicked(java.awt.event.MouseEvent evt) {
    new findRecord().setVisible(true);
}
```

Run the program. Check that '***Add record***' and '***Find record***' windows can be opened by clicking the menu items. Check also that these windows can be closed without exiting from the main program.

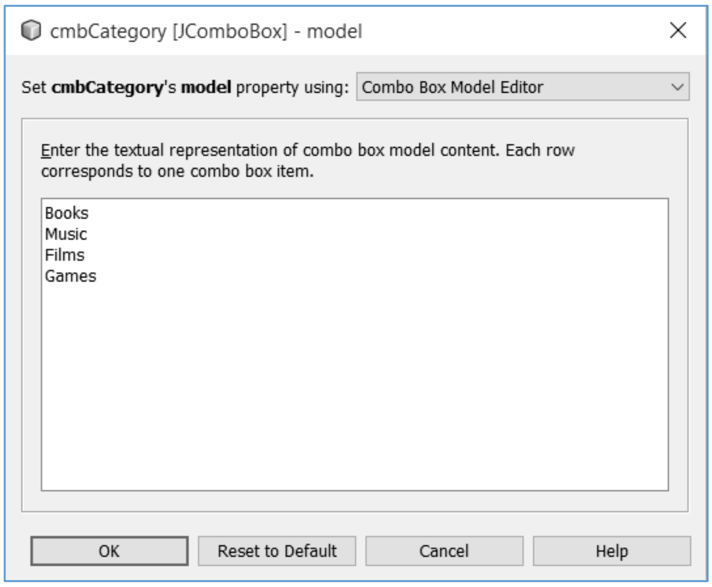


Close the program and return to the NetBeans editing screen. Use the tab to move to the ***addRecord.java*** page. Add components to the form to allow product records to be input:

- A label with the caption '***Add record***'
- A label with the caption '***Product code***'. Place a text field alongside and rename this as ***txtProductCode***.
- A label with the caption '***Category***'. Place a Combo Box alongside and rename this as ***cmbCategory***.
- A label with the caption '***Title / description***'. Place a text field alongside and rename this as ***txtDescription***.
- A button with the caption '***Add record***'. Rename the button as ***btnAdd***.



Select the ***cmbCategory*** combo box. Go to the Properties window and locate the ***model*** property. Click the ***ellipsis*** (…) symbol at the end of the row to open an editing window. Add the items for the drop down list: ***Books***, ***Music***, ***Films*** and ***Games***.


cmbCategory [JComboBox] - model

Set **cmbCategory**'s **model** property using: Combo Box Model Editor

Enter the textual representation of combo box model content. Each row corresponds to one combo box item.

```
Books
Music
Films
Games
```

Click the ***OK*** button to return to the form layout view.

Use the ***Source*** tab to move to the program code screen. Add Java the modules at the start of the program listing which will be needed for file handling and to produce a message box.

```
package randomAccessPackage;

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

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

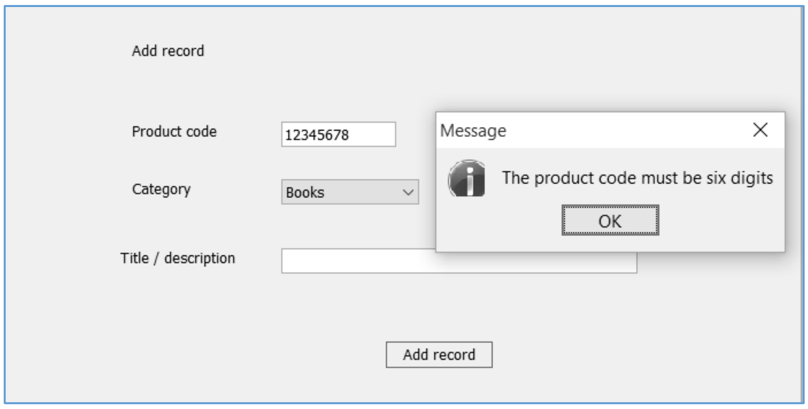
Click the ***Design*** tab to return to the form layout view. Double click the '***Add record***' button to create a method. Add the line of code to call an ***addRecord( )*** method, then begin the method immediately underneath. We will begin by checking that the ***product code*** entered has a correct length of ***six characters***. Please note that the line beginning

'JOptionPane.showMessageDialog(…'

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

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

    addRecord();

}

private void addRecord()
{
    String productCodeEntered=txtProductCode.getText().trim();
    if (productCodeEntered.length()!=6)
    {
        JOptionPane.showMessageDialog(addRecord.this,
                                    "The product code must be six digits");
    }
}
```

Run the program. Select the '***Add record***' menu option. Check that an error message appears if the Product code entered does not have a length of six characters.



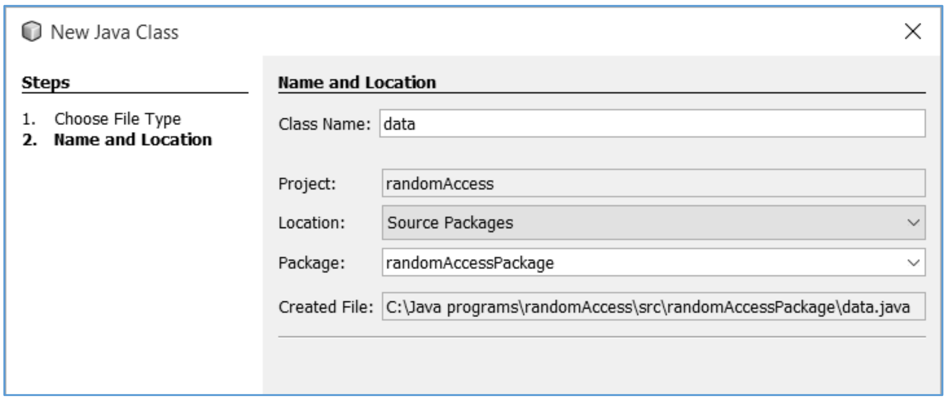
12345678

The product code must be six digits

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

Before developing the program further, we will set up a ***data*** class file to store global variables which will be needed by several of the program forms.

Locate the ***randomAccessPackage*** folder in the Project window at the top left of the screen. Right-click on ***randomAccessPackage*** and select ***New / Java Class***. Set the ***Class Name*** as ***data***. Leave the ***Package*** name as ***randomAccessPackage***.


New Java Class

Steps
1. Choose File Type
2. Name and Location

Name and Location

Class Name: data

Project: randomAccess

Location: Source Packages

Package: randomAccessPackage

Created File: C:\Java programs\randomAccess\src\randomAccessPackage\data.java

Click the ***Finish*** button to open the ***data*** class file. Add a variable ***fileLocations*** which will record the number of record storage locations which will be allocated on disc for the random access file. We will also assign a name for the disc file.

```
package randomAccessPackage;

public class data {

    public static int fileLocations;
    public static String filename = "randomAccess.dat";

}
```

When the program begins, we must initialise the number of storage locations in the random access file. Use the tab above the editing screen to move to the ***randomAccess.java*** page. Add a line of code to the ***randomAccess( )*** method to set the initial number of storage locations to 10.

```
package randomAccessPackage;

public class randomAccess extends javax.swing.JFrame {

    public randomAccess() {
        initComponents();

        data.fileLocations=10;

    }
```

Use the tab to return to the ***addRecord.java*** page.

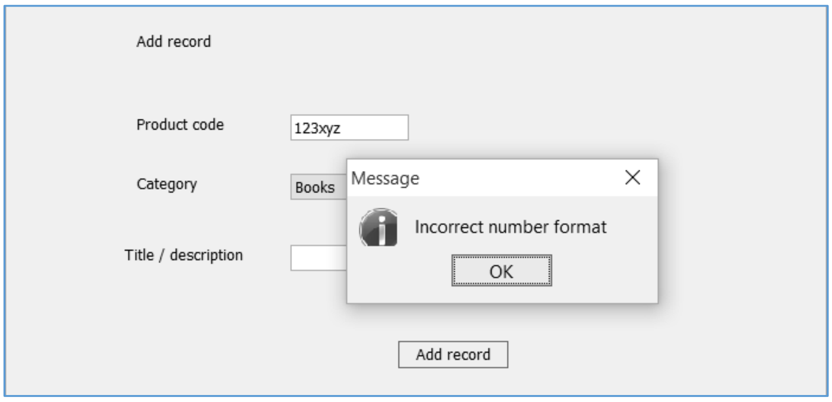
Locate the ***addRecord( )*** method which you were developing earlier. We will now use the product code to calculate the file location where the record should be stored, using the formula:

<product code> MOD <number of file locations>

Add a call to a function ***getHashValue( )*** which will carry out the calculation and return the number of the file location where the record should be stored. This function should be inserted immediately below the ***addRecord( )*** method. The line beginning '***JOptionPane.showMessageDialog(…***' should be entered as a single line of code with no line break.

```
        private void addRecord()
        {

            String productCodeEntered=txtProductCode.getText().trim();
            if (productCodeEntered.length()!=6)
            {
                 JOptionPane.showMessageDialog(addRecord.this,
                                      "The product code must be six digits");
            }
            else
            {
                int n=getHashValue(productCodeEntered);
            }
        }
        private int getHashValue(String productCode)
        {
            int hashValue=-1;
            try
            {
                long code=Long.parseLong(productCode);
                hashValue = (int) (code % data.fileLocations);
            }
            catch (NumberFormatException e)
            {
                JOptionPane.showMessageDialog(addRecord.this,
                                                   "Incorrect number format");
            }
            return hashValue;
        }
```

We have included error trapping, in case the product code which is entered contains non-numeric characters. Run the program. Go to the '***Add record***' page. Enter a product code containing one or more letters, and check that the error is detected.


Add record

Product code: 123xyz

Category: Books

Title / description

Message

Incorrect number format

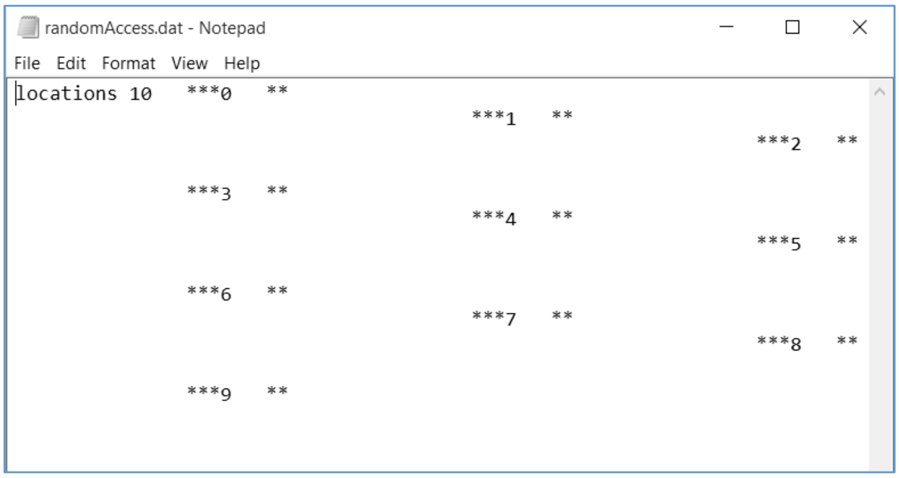
Close the program windows and return to the NetBeans editing screen.

Before saving the record, we must set up the structure on disc for the random access file. Use the tab above the editing screen to move to the ***randomAccess.java*** page. We will add sections of code:

- At the start of the program listing, include the Java modules which will be needed for file handling.
- Add a block of code to the ***randomAccess( )*** method which will check whether the random access file already exists on the disc. If not, it calls a method to create the file.
- Include the ***createFile( )*** method immediately underneath the randomAccess( ) method. The createFile( ) method uses a loop to create the required number of blank storage locations. We are allowing a fixed length of ***100 bytes*** for each storage location.

```
package randomAccessPackage;

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

public class randomAccess extends javax.swing.JFrame {

    public randomAccess() {
        initComponents();
        data.fileLocations=10;

        File f = new File(data.filename);
        if(f.exists()==false)
        {
            createFile();
        }

    }

    private void createFile()
    {
        try (RandomAccessFile file = new RandomAccessFile(data.filename, "rw"))
        {
            data.fileLocations=10;
            String s = "locations "+data.fileLocations;
            file.write(s.getBytes());
            for (int i=0; i<data.fileLocations; i++)
            {
               String locationNumber=String.format("%-4s", i);
               s = "***"+locationNumber+"** ";
               int position=i*100+15;
               file.seek(position);
               file.write(s.getBytes());
            }
            file.close();
        }
        catch(IOException e)
        {
              JOptionPane.showMessageDialog(randomAccess.this, "File error");
        }
    }
```

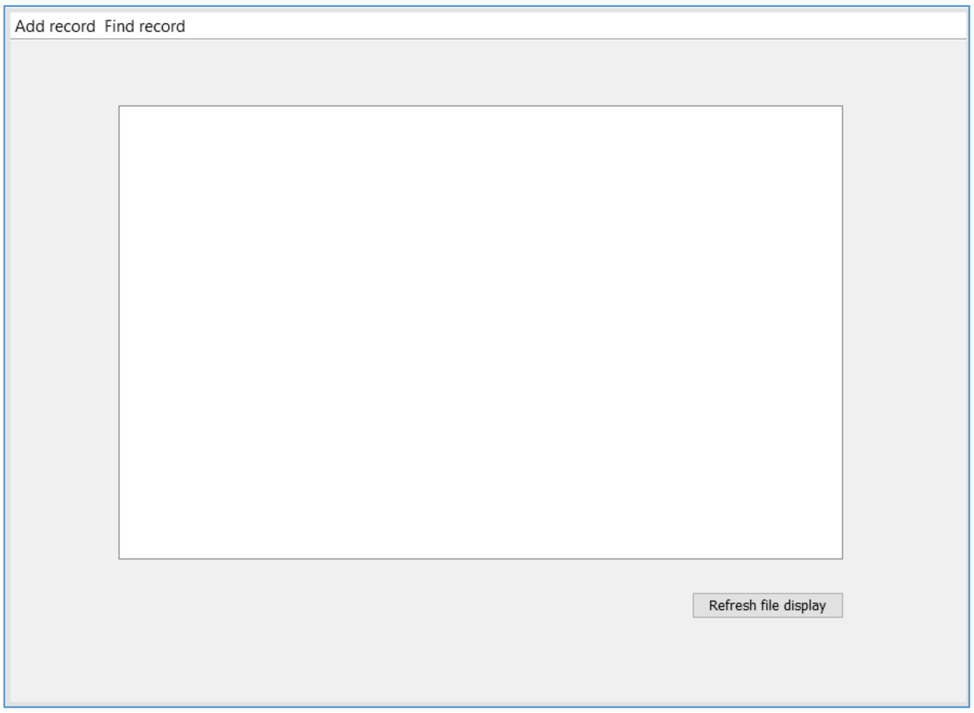
Run the program. Use Windows Explorer to locate the file randomAccess.dat in the ***randomAccess*** project folder. Open the file with a text editing application such as Notepad. Check that the file contains a message giving the number of storage locations, followed by the correct number of blank fixed length records.



```
locations 10    ***0    **
                                        ***1    **
                                                                        ***2    **
                ***3    **
                                        ***4    **
                                                                        ***5    **
                ***6    **
                                        ***7    **
                                                                        ***8    **
                ***9    **
```

Close the program and return to the NetBeans editing screen. As we test the random access file, it will be more convenient if the contents of the file are displayed on the program screen, rather than having to keep opening the file in a separate application. We will set up a display method on the main program page.

Use the ***Design*** tab to move to the form layout view. Add a ***text area*** component, giving this the name ***txtOutput***. Insert a button below the text area, with the caption '***Refresh file display***'. Rename the button as ***btnRefresh***.



Double click the '***Refresh file display***' button to create a method.

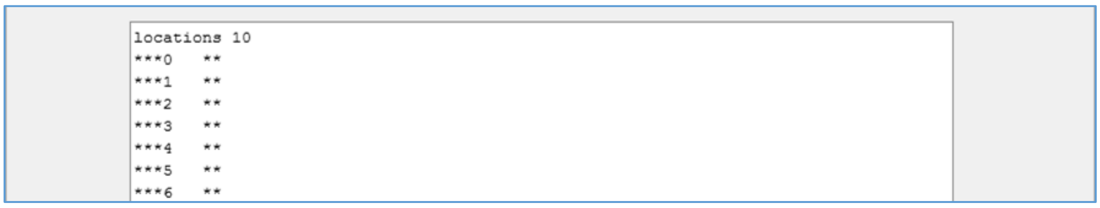
Add a line of code to the button click method to call a ***displayFile( )*** method. Insert ***displayFile( )*** immediately below the button click method. This method uses a loop to access each of the records from the random access file, then adds each record to the text string which is output.

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

private void displayFile()
{
    int position;
    String output="";
    byte[] bytes;
    try
    {
        RandomAccessFile file = new RandomAccessFile(data.filename, "r");
        bytes = new byte[15];
        file.read(bytes);
        String s=new String(bytes);
        output += s+"\n";
        s=s.substring(10);
        data.fileLocations=Integer.parseInt(s.trim());
        for (int i=0; i<data.fileLocations; i++)
        {
            position=i*100 + 15;
            file.seek(position);
            bytes = new byte[100];
            file.read(bytes);
            s=new String(bytes);
            output += s+"\n";
        }
        file.close();
        txtOutput.setText(output);
    }
    catch(IOException e)
    {
        JOptionPane.showMessageDialog(randomAccess.this, "File error");
    }
}
```

Scroll up to the top of the program listing and add a line at the end of the ***randomAccess( )*** method to call the ***displayFile( )*** method when the program first runs.

```
public randomAccess() {
    initComponents();
    data.fileLocations=10;
    File f = new File(data.filename);
    if(f.exists()==false)
    {
        createFile();
    }
    f = new File(data.overflow);
    if(f.exists()==false)
    {
        createOverflow();
    }
    displayFile();
}
```

Run the program. Check that the set of blank records is displayed in the text area. Notice that the locations are numbered 0 to 9, which are the possible remainder values when a product code is divided by 10.



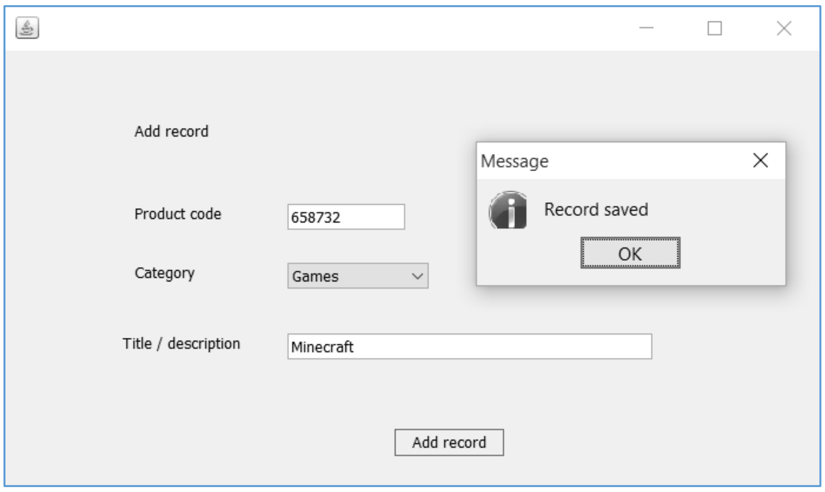
```
locations 10
***0   **
***1   **
***2   **
***3   **
***4   **
***5   **
***6   **
```

Close the program window and return to the NetBeans editing screen. We are now ready to save product records into the file. Click the tab to return to the ***addRecord.java*** page and locate the ***addRecord( )*** method. We will add code to carry out a series of tasks:

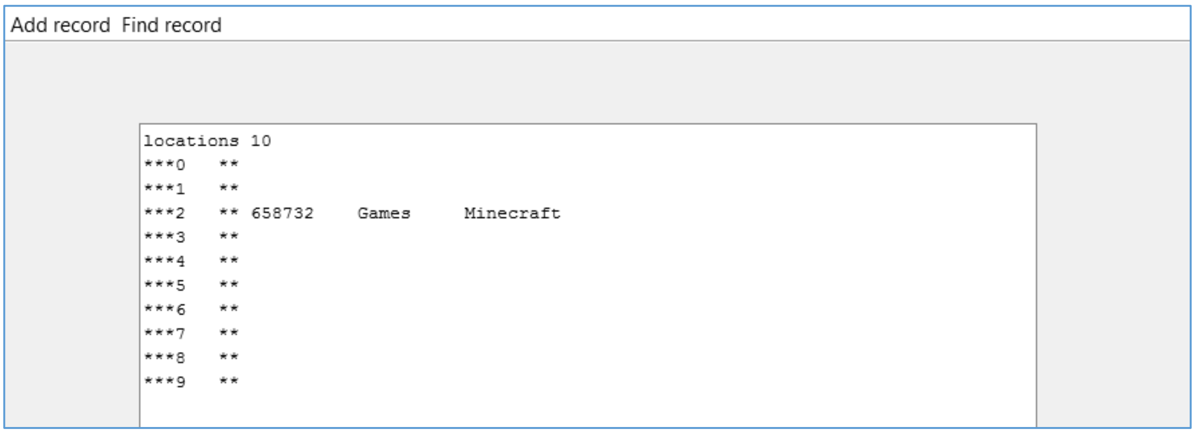
- We check to make sure that a valid product code has been entered, and a hash value ***n*** has been calculated.
- The product data is assembled into a fixed length record. The ***product code*** is given a field length of ***10 bytes***, the ***category*** is ***10 bytes*** and the ***title***/***description*** is ***70 bytes***.
- The file location ***n*** is used to calculate the position in the file where the record will be stored.

```
private void addRecord()
{
    String productCodeEntered=txtProductCode.getText().trim();
    if (productCodeEntered.length()!=6)
    {
         JOptionPane.showMessageDialog(addRecord.this,
                                   "The product code must be six digits");
    }
    else
    {
       int n=getHashValue(productCodeEntered);

       if (n>=0)
       {
          int position;
          String category =(String) cmbCategory.getSelectedItem();
          String description=txtDescription.getText();
          productCodeEntered=String.format("%-10s", productCodeEntered);
          category=String.format("%-10s", category);
          description=String.format("%-70s", description);
          String productRecord=productCodeEntered+category+description;
          productRecord=productRecord.substring(0, 90);
          try(RandomAccessFile file = new RandomAccessFile(data.filename, "rw"))
          {
              position=100*n + 25;
              file.seek(position);
              file.write(productRecord.getBytes());
              file.close();
              JOptionPane.showMessageDialog(addRecord.this, "Record saved");
          }
          catch(IOException e)
          {
              JOptionPane.showMessageDialog(addRecord.this, "File error");
          }
          txtProductCode.setText("");
          txtDescription.setText("");
       }
   }
}
```

Run the program. Select the '***Add record***' menu option. Enter the details of a product sold by the on-line store, then click the '***Add record***' button.

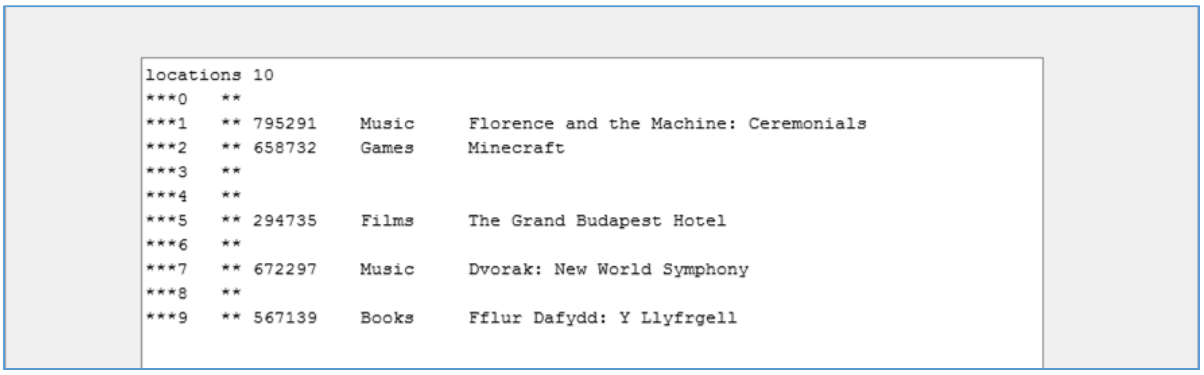


Move to the main program window and click the '***Refresh file display***' button. Check that the record which you have entered is displayed in the correct storage location, representing the remainder when the product code is divided by 10.



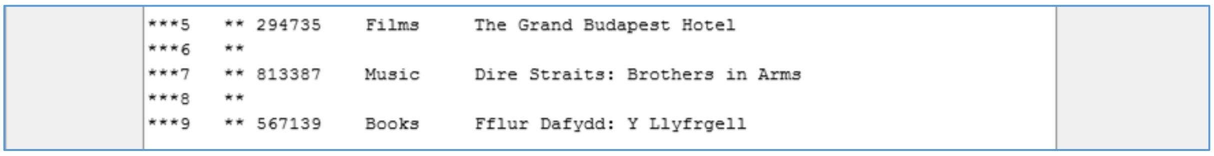
```
locations 10
***0   **
***1   **
***2   ** 658732    Games     Minecraft
***3   **
***4   **
***5   **
***6   **
***7   **
***8   **
***9   **
```

Enter further records, each time refreshing the file display.



```
locations 10
***0   **
***1   ** 795291    Music     Florence and the Machine: Ceremonials
***2   ** 658732    Games     Minecraft
***3   **
***4   **
***5   ** 294735    Films     The Grand Budapest Hotel
***6   **
***7   ** 672297    Music     Dvorak: New World Symphony
***8   **
***9   ** 567139    Books     Fflur Dafydd: Y Llyfrgell
```

All seems to go well until a record is entered which generates the same hash value as an exiting record in the file, as in the case of ***672297*** and ***813387***. The earlier record is then overwritten.



```
***5   ** 294735    Films     The Grand Budapest Hotel
***6   **
***7   ** 813387    Music     Dire Straits: Brothers in Arms
***8   **
***9   ** 567139    Books     Fflur Dafydd: Y Llyfrgell
```

Close the program windows and return to the NetBeans editing screen. We must now develop a strategy to handle collisions. A simple solution is to provide an unsorted overflow file where records can be stored if the required location in the main file is already occupied.

Use the tab above the editing window to move to the ***data.java*** class file. Add a name for the overflow file.

```
public class data {

   public static int fileLocations;
   public static String filename = "randomAccess.dat";

   public static String overflow = "overflow.dat";

}
```

Click the tab to move to the ***randomAccess.java*** page. Add lines of code to the ***randomAccess( )*** method which will check whether an overflow file already exists. If not, a method ***createOverflow( )*** will be called. Insert this method immediately below the ***randomAccess( )*** method.

```
    public randomAccess() {
        initComponents();
        data.fileLocations=10;
        File f = new File(data.filename);
        if(f.exists()==false)
        {
            createFile();
        }

        f = new File(data.overflow);
        if(f.exists()==false)
        {
            createOverflow();
        }

    }

    private void createOverflow()
    {
        try (RandomAccessFile file = new RandomAccessFile(data.overflow, "rw"))
        {
            file.setLength(0);
            file.close();
        }
        catch(IOException e)
        {
               JOptionPane.showMessageDialog(randomAccess.this, "File error");
        }
    }
```

Run the program. Use Windows Explorer to check that an empty ***overflow.dat*** file has been created in the ***randomAccess*** project folder.

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

We can now work on the storage of records in the overflow area when collisions occur.

Click the tab above the editing screen to return to the ***addRecord.java*** page. Locate the ***addRecord( )*** method.

We will need to add extra code to fully implement our strategy for handling collisions:

- Input the product code
- Use the hash function to calculate the storage location in the main file
- Check whether this location is already occupied
- IF location is not occupied THEN store the new record in the main file
- ELSE store the new record in the overflow area.

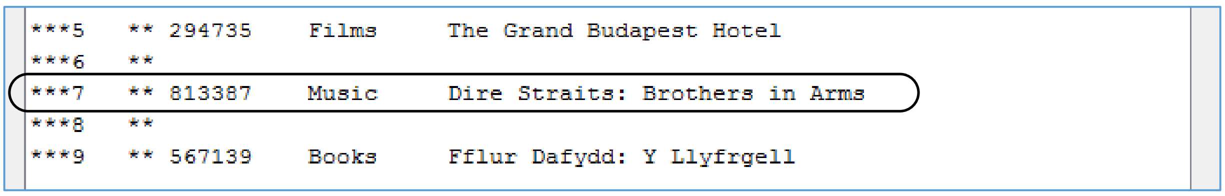
We will firstly add the lines of code necessary to check whether the required location in the main file is already occupied. The program will open the random access file and return any product code found at that location.

```
private void addRecord()
{
    String productCodeEntered=txtProductCode.getText().trim();
    if (productCodeEntered.length()!=6)
    {
         JOptionPane.showMessageDialog(addRecord.this,
                                      "The product code must be six digits");
    }
    else
    {
       int n=getHashValue(productCodeEntered);
       if (n>=0)
       {
          int position;
          String category =(String) cmbCategory.getSelectedItem();
          String description=txtDescription.getText();

          String s="";
          String productCodeFound="";
          byte[] bytes;
          try(RandomAccessFile file = new RandomAccessFile(data.filename, "rw"))
          {
              position=100*n + 25;
              file.seek(position);
              bytes = new byte[10];
              file.read(bytes);
              file.close();
              s= new String (bytes);
              productCodeFound=s.trim();
           }
           catch(IOException e)
           {
              JOptionPane.showMessageDialog(addRecord.this, "File error");
           }

           productCodeEntered=String.format("%-10s", productCodeEntered);
           category=String.format("%-10s", category);
           description=String.format("%-70s", description);
```

We will now check whether a product code is already present at the calculated file location. If not, then the code written earlier will be used to store the new record in the main file. However, if the location is already occupied then the record will be added instead to the overflow file.

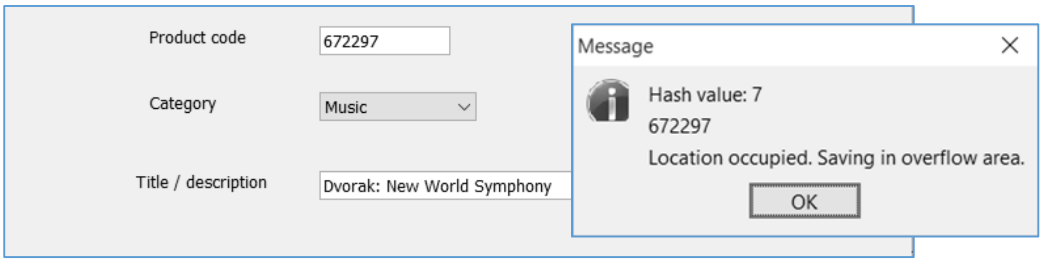
```
        productCodeEntered=String.format("%-10s", productCodeEntered);
        category=String.format("%-10s", category);
        description=String.format("%-70s", description);
        String productRecord=productCodeEntered+category+description;
        productRecord=productRecord.substring(0, 90);

        if (productCodeFound.length()<1)
        {

            try(RandomAccessFile file = new RandomAccessFile(data.filename, "rw"))
            {
                position=100*n + 25;
                file.seek(position);
                file.write(productRecord.getBytes());
                file.close();
                JOptionPane.showMessageDialog(addRecord.this, "Record saved");
            }
            catch(IOException e)
            {
                 JOptionPane.showMessageDialog(addRecord.this, "File error");
            }

        }
        else
        {
            String message ="Hash value: "+n+"\n"+productCodeEntered;
            message+="\nLocation occupied. Saving in overflow area.";
            JOptionPane.showMessageDialog(addRecord.this,message);
            try(RandomAccessFile file = new RandomAccessFile(data.overflow, "rw"))
            {
                 position=(int) file.length();
                 file.seek(position);
                 file.write(productRecord.getBytes());
                 file.close();
            }
            catch(IOException e)
            {
                JOptionPane.showMessageDialog(addRecord.this, "File error");
            }
        }

        txtProductCode.setText("");
        txtDescription.setText("");
    }
```

Run the program. Click the '***Refresh file display***' button to view the records in the random access file. Select one of the current records:



```
***5   ** 294735    Films     The Grand Budapest Hotel
***6   **
***7   ** 813387    Music     Dire Straits: Brothers in Arms
***8   **
***9   ** 567139    Books     Fflur Dafydd: Y Llyfrgell
```

Devise a product code which will generate the same hash value. Click the '***Add record***' menu option, then enter a product with this code.



Product code: 672297

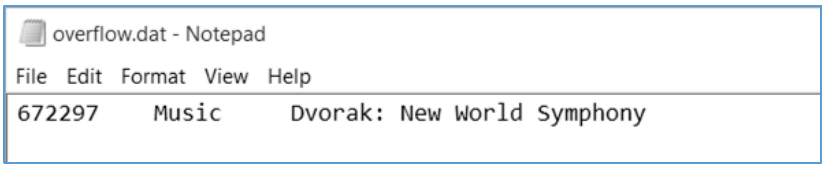
Category: Music

Title / description: Dvorak: New World Symphony

Message

Hash value: 7
672297
Location occupied. Saving in overflow area.

Use Windows Explorer to locate the ***overflow.dat*** file in the ***randomAccess*** project folder, then open the file with a text editor. Check that the additional record has been saved correctly.



overflow.dat - Notepad

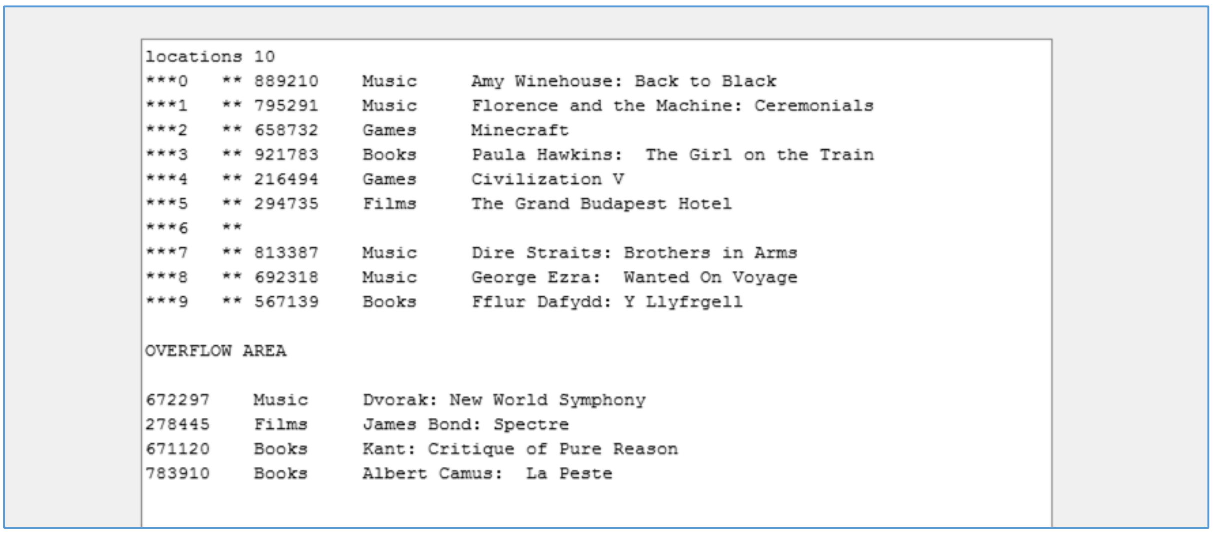
```
672297    Music     Dvorak: New World Symphony
```

Close the program windows and return to the NetBeans editing screen. As with the main file, it would be helpful for testing the project if the overflow file is displayed on screen when the program is running. We will arrange for this to happen.

Use the tab above the editing screen to move to the ***randomAccess.java*** page. Locate the ***displayFile( )*** method, and add a block of code to also output the contents of the ***overflow.dat*** file.

```
        for (int i=0; i<data.fileLocations; i++)
        {
            position=i*100 + 15;
            file.seek(position);
            bytes = new byte[100];
            file.read(bytes);
            s=new String(bytes);
            output += s+"\n";
        }
        file.close();

        output += "\nOVERFLOW AREA\n\n";
        file = new RandomAccessFile(data.overflow, "r");
        int overflowRecords=(int) (file.length()/90);
        for (int i=0; i<overflowRecords; i++)
        {
            position=i*90;
            file.seek(position);
            bytes = new byte[90];
            file.read(bytes);
            s=new String(bytes);
            output += s+"\n";
        }
        file.close();

        txtOutput.setText(output);
    }
    catch(IOException e)
    {
        JOptionPane.showMessageDialog(randomAccess.this, "File error");
    }
```

Run the program. Check that records in both the main file and the overflow area are displayed correctly. Open the '***Add record***' window and add further products. Check that each record is correctly stored in either the calculated location in the main file, or in the overflow area if a collision has occurred. Enlarge the text area if necessary, so that the complete records are visible.

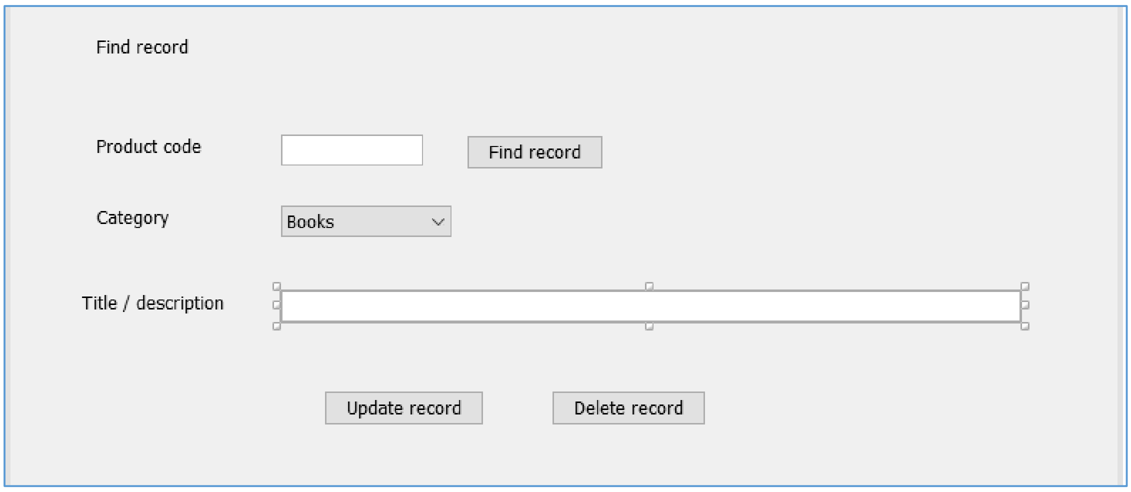


```
locations 10
***0   ** 889210    Music     Amy Winehouse: Back to Black
***1   ** 795291    Music     Florence and the Machine: Ceremonials
***2   ** 658732    Games     Minecraft
***3   ** 921783    Books     Paula Hawkins:  The Girl on the Train
***4   ** 216494    Games     Civilization V
***5   ** 294735    Films     The Grand Budapest Hotel
***6   **
***7   ** 813387    Music     Dire Straits: Brothers in Arms
***8   ** 692318    Music     George Ezra:  Wanted On Voyage
***9   ** 567139    Books     Fflur Dafydd: Y Llyfrgell

OVERFLOW AREA

672297    Music     Dvorak: New World Symphony
278445    Films     James Bond: Spectre
671120    Books     Kant: Critique of Pure Reason
783910    Books     Albert Camus:  La Peste
```

Close the program windows and return to the NetBeans editing screen. We will now work on a page to find and display individual records, with options to edit or delete the record.

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

- A label with the caption '***Find record***'
- A label with the caption '***Product code***'. Place a text field alongside and rename this as ***txtProductCode***. Also add a button with the caption '***Find record***'. Rename the button as ***btnFind***.
- A label with the caption '***Category***'. Place a Combo Box alongside and rename this as ***cmbCategory***. Set up the categories: ***Books***, ***Music***, ***Films*** and ***Games***, as on the ***addRecord.java*** page.
- A label with the caption '***Title / description***'. Place a text field alongside and rename this as ***txtDescription***.
- 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 view. Add the Java modules at the start of the program which are needed for file handling, and global variables which will be used in the program.

```
package randomAccessPackage;

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

public class findRecord extends javax.swing.JFrame {

    int n;
    Boolean found;
    String productCodeWanted;
    int overflowLocation;

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

Use the ***Design*** tab to return to the form layout view, then double click the '***Find record***' button to create a method. Add a line of code to open a ***find( )*** method, then add the ***find( )*** method immediately underneath the button click method.

The method begins by calculating a hash value from the product code entered for the search. If the product code is valid, the random access file will be opened.

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

  find();

}

private void find()
{
    found=false;
    int position;
    txtDescription.setText("");
    try
    {
        productCodeWanted=txtProductCode.getText();
        n=getHashValue(productCodeWanted);
        if (n<0)
        {
            JOptionPane.showMessageDialog(findRecord.this,
                                                 "Incorrect product code");
        }
        else
        {
            RandomAccessFile file = new RandomAccessFile(data.filename, "r");
        }
    }
    catch(IOException e)
    {
        JOptionPane.showMessageDialog(findRecord.this, "File error");
    }
 }
```

The ***find( )*** method will require a function to calculate the hash value from the product code. Insert this function below the ***find( )*** method.

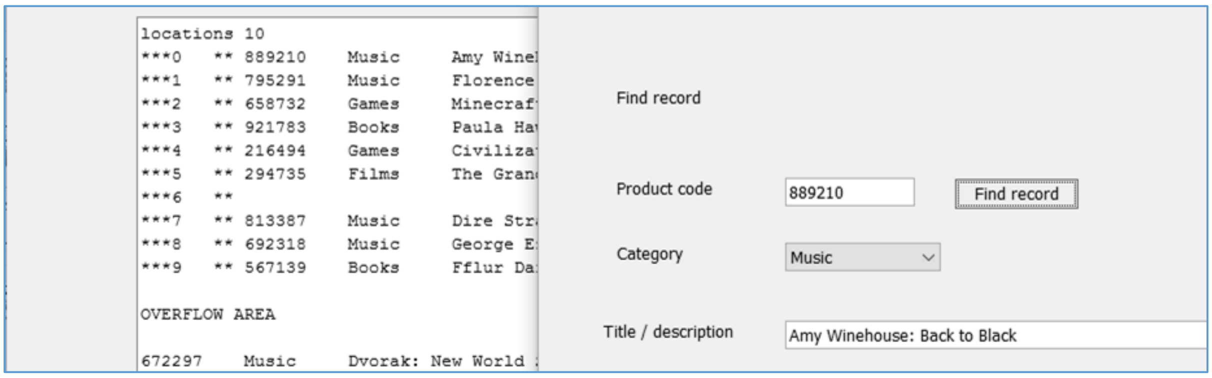
```
private int getHashValue(String productCode)
{
    int hashValue=-1;
    try
    {
        long code=Long.parseLong(productCode);
        hashValue = (int) (code % data.fileLocations);
    }
    catch (NumberFormatException e)
    {
        JOptionPane.showMessageDialog(findRecord.this, "Incorrect number format");
    }
    return hashValue;
}
```

Return to the ***find( )*** method and add lines of code which will carry out several tasks:

- The hash value n is used to calculate the position of the required record in the file.
- The record at the calculated position is loaded, and the record is split into the ***productCode***, ***category*** and ***description*** fields.
- If the product code matches the product code entered for the search, then the record is displayed. However, the required record may not have been found. It may either be in the overflow area due to a collision, or not present at all.

```
        if (n<0)
        {
            JOptionPane.showMessageDialog(findRecord.this,
                                                 "Incorrect product code");
        }
        else
        {
            RandomAccessFile file = new RandomAccessFile(data.filename, "r");
            position=100*n+25;
            file.seek(position);
            byte[] bytes = new byte[90];
            file.read(bytes);
            file.close();
            String s=new String(bytes);
            String productCode=s.substring(0,10); s=s.substring(10);
            String category=s.substring(0,10); s=s.substring(10);
            String description=s.substring(0,70);
            if(productCodeWanted.trim().equals(productCode.trim()))
            {
                found=true;
                txtDescription.setText(description);
                cmbCategory.setSelectedItem(category.trim());
            }
        }
    }
    catch(IOException e)
```

Run the program. Click the menu to open the '***Find record***' page. Enter the product codes for some items stored in the ***main file***. Details should be displayed; enlarge the text field if necessary so that the complete title/description is visible. Please note, however, that records stored in the ***overflow area*** are not yet displayed when the product codes are entered. We will correct this problem next.



```
locations 10
***0    ** 889210    Music      Amy Winel
***1    ** 795291    Music      Florence
***2    ** 658732    Games      Minecraf
***3    ** 921783    Books      Paula Ha
***4    ** 216494    Games      Civiliza
***5    ** 294735    Films      The Gran
***6    **
***7    ** 813387    Music      Dire Str
***8    ** 692318    Music      George E
***9    ** 567139    Books      Fflur Da

OVERFLOW AREA

672297    Music      Dvorak: New World
```

Find record

Product code: 889210

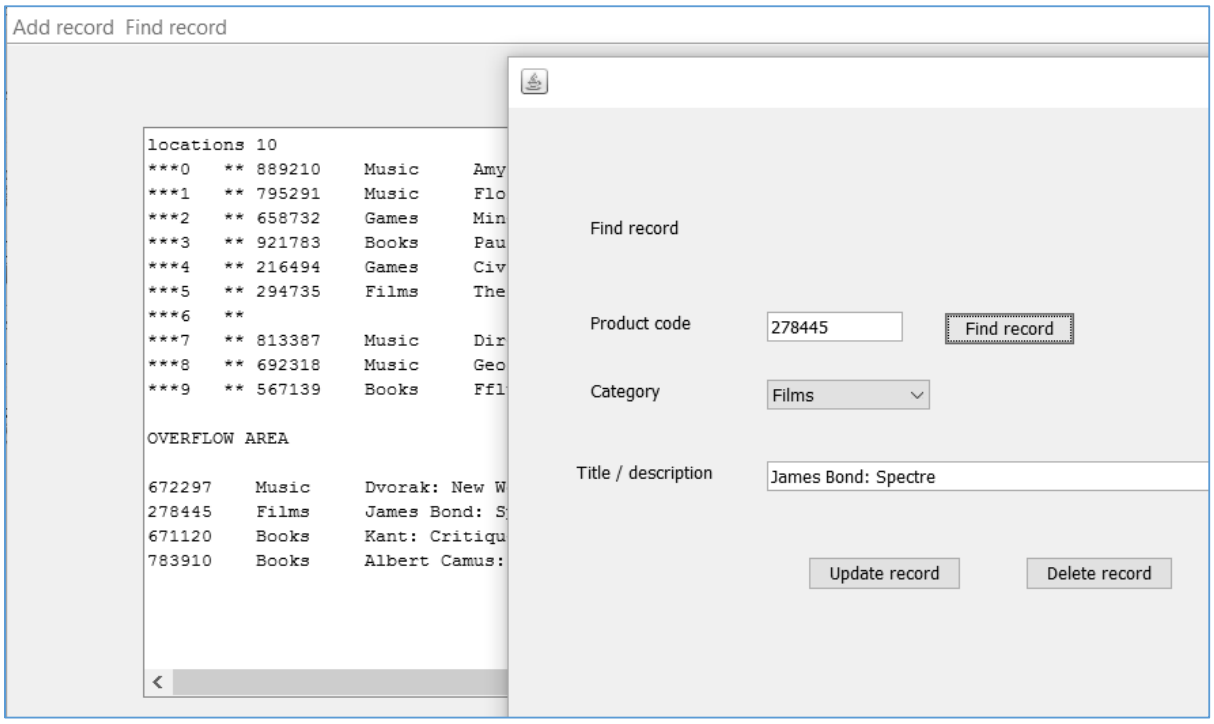
Category: Music

Title / description: Amy Winehouse: Back to Black

Close the program windows and return to the editing screen. Return to the ***find( )*** method and add lines of code which will search the overflow area for the required product code, then display the record if it is found.

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

        overflowLocation=-1;
        if (found==false)
        {
             try
             {
                    RandomAccessFile file = new RandomAccessFile(data.overflow, "r");
                    int recordCount=(int) file.length()/90;
                    int i=0;
                    while(found==false && i<recordCount)
                    {
                        position=90*i;
                        file.seek(position);
                        byte[] bytes = new byte[90];
                        file.read(bytes);
                        String s=new String(bytes);
                        String productCode=s.substring(0,10); s=s.substring(10);
                        String category=s.substring(0,10); s=s.substring(10);
                        String description=s.substring(0,70);
                        if(productCodeWanted.trim().equals(productCode.trim()))
                        {
                             found=true;
                             txtDescription.setText(description);
                             cmbCategory.setSelectedItem(category.trim());
                             overflowLocation=i;
                        }
                        i++;
                    }
                    file.close();
             }
             catch(IOException e)
             {
                 JOptionPane.showMessageDialog(findRecord.this, "File error");
             }
        }
    }
```

Run the program. Repeat the previous tests, and check that it is now possible to find and display records from both the ***main file*** and the ***overflow area***.



```
locations 10
***0    ** 889210    Music      Amy
***1    ** 795291    Music      Flo
***2    ** 658732    Games      Min
***3    ** 921783    Books      Pau
***4    ** 216494    Games      Civ
***5    ** 294735    Films      The
***6    **
***7    ** 813387    Music      Dir
***8    ** 692318    Music      Geo
***9    ** 567139    Books      Ffl

OVERFLOW AREA

672297    Music      Dvorak: New W
278445    Films      James Bond: S
671120    Books      Kant: Critiqu
783910    Books      Albert Camus:
```

Product code: 278445
Category: Films
Title / description: James Bond: Spectre

Close the program windows and return to the NetBeans editing screen. We will work next on the option to update a record.

Use the ***Design*** tab to move to the form layout view, then double click the '***Update record***' button to produce a button click method. Add a line of code to call an ***update( )*** method. Create the update( ) method immediately underneath, and add lines of code which will collect the data entries for the updated record and package these into the correct fixed length record format.

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

    private void update()
    {
        String category =(String) cmbCategory.getSelectedItem();
        String description=txtDescription.getText();
        String productCode=String.format("%-10s", productCodeWanted);
        category=String.format("%-10s", category);
        description=String.format("%-70s", description);
        String s = productCode+ category + description;
        s=s.substring(0, 90);
    }
```

As in the ***find( )*** method, we will need to treat records differently for the main file and overflow area when carrying out the update. When the update method is called, the values of two global variables have already been set:

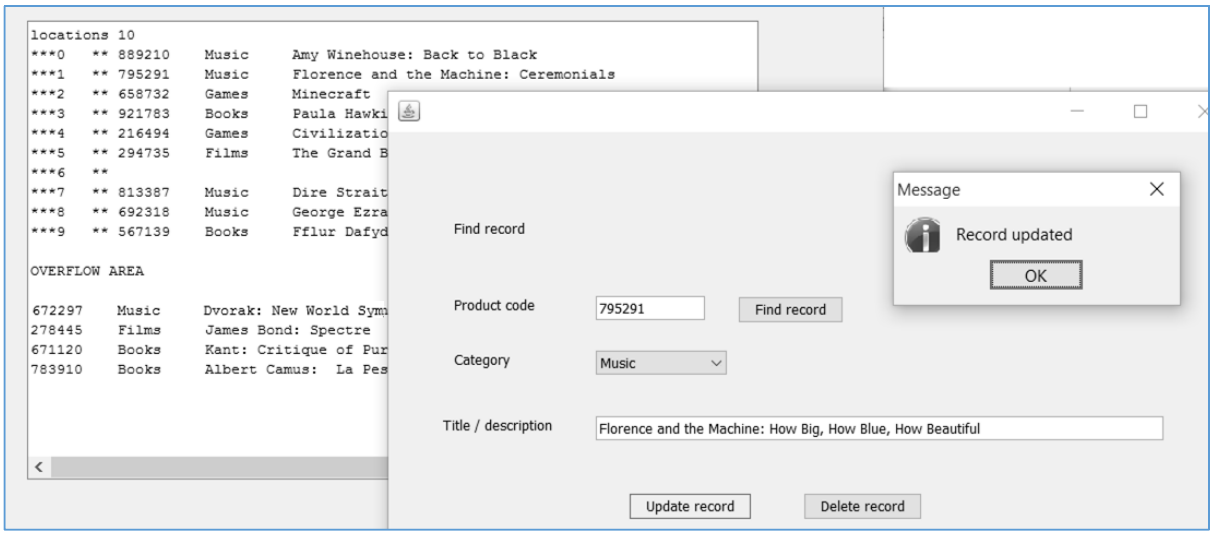
n: the hash value for the location where the record would be stored if it is in the main file.

overflowLocation: gives the position of the record in the overflow file if it is stored there. However, this variable will have a value of -1 if the record is not stored in the overflow area, but stored instead in the main file.

We will first deal with the case where the record is in the main file, and ***overflowLocation*** has a value of -1. Add lines of code to update the main file entry.

```
    private void update()
    {
        String category =(String) cmbCategory.getSelectedItem();
        String description=txtDescription.getText();
        String productCode=String.format("%-10s", productCodeWanted);
        category=String.format("%-10s", category);
        description=String.format("%-70s", description);
        String s = productCode+ category + description;
        s=s.substring(0, 90);

        if (overflowLocation<0)
        {
            try(RandomAccessFile file = new RandomAccessFile(data.filename, "rw"))
            {
                int position=100*n + 25;
                file.seek(position);
                file.write(s.getBytes());
                file.close();
                JOptionPane.showMessageDialog(findRecord.this, "Record updated");
            }
            catch(IOException e)
            {
                JOptionPane.showMessageDialog(findRecord.this, "File error");
            }
        }
        txtProductCode.setText("");
        txtDescription.setText("");

    }
```

Run the program. Go to the '***Find record***' page, then make a change to the ***title / description*** field of one of the products in the ***main file***. Check that this is updated correctly.



```
locations 10
***0   ** 889210    Music     Amy Winehouse: Back to Black
***1   ** 795291    Music     Florence and the Machine: Ceremonials
***2   ** 658732    Games     Minecraft
***3   ** 921783    Books     Paula Hawki
***4   ** 216494    Games     Civilizatio
***5   ** 294735    Films     The Grand B
***6   **
***7   ** 813387    Music     Dire Strait
***8   ** 692318    Music     George Ezra
***9   ** 567139    Books     Fflur Dafyd

OVERFLOW AREA

672297    Music     Dvorak: New World Sym
278445    Films     James Bond: Spectre
671120    Books     Kant: Critique of Pur
783910    Books     Albert Camus:  La Pes
```

Find record

Product code: 795291

Category: Music

Title / description: Florence and the Machine: How Big, How Blue, How Beautiful

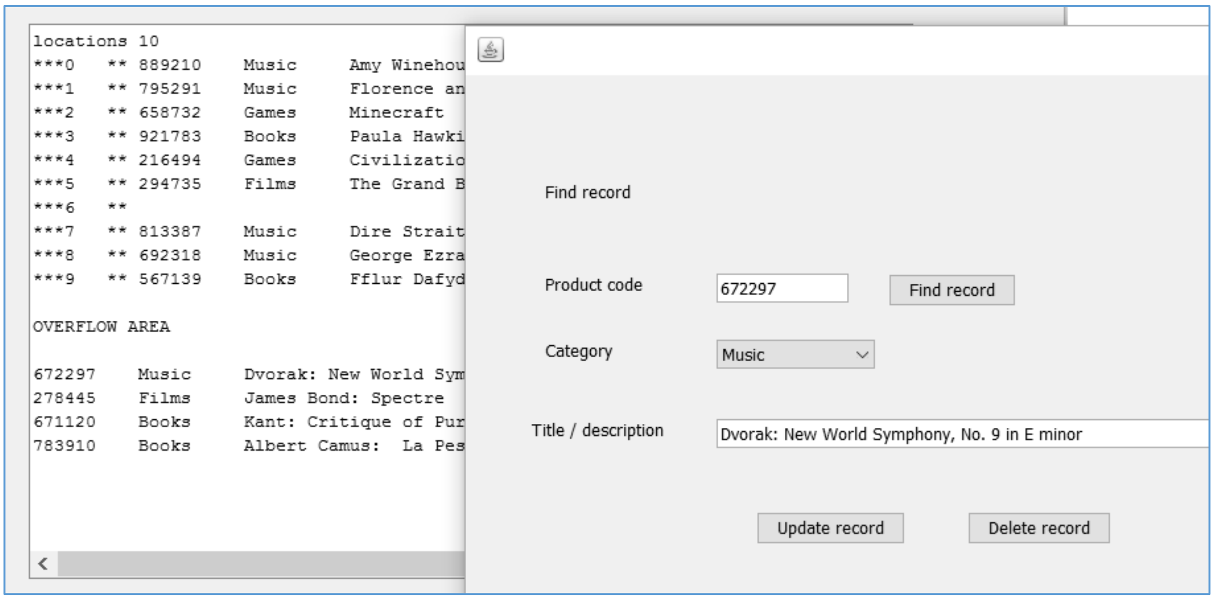
Message

Record updated

Close the program windows and return to the editing screen. Locate the ***update( )*** method. We will now insert the lines of code needed to update records in the ***overflow area***.

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

        }
        else
        {
            try(RandomAccessFile file = new RandomAccessFile(data.overflow, "rw"))
            {
                int position=90* overflowLocation;
                file.seek(position);
                file.write(s.getBytes());
                file.close();
                JOptionPane.showMessageDialog(findRecord.this, "Record updated");
            }
            catch(IOException e)
            {
                JOptionPane.showMessageDialog(findRecord.this, "File error");
            }
        }
        txtProductCode.setText("");
        txtDescription.setText("");
    }
```

Run the program. Go to the '***Find record***' page and repeat the test of the update method, this time choosing a record stored in the ***overflow area***. Check that the record is updated correctly.



```
locations 10
***0   ** 889210    Music     Amy Winehou
***1   ** 795291    Music     Florence an
***2   ** 658732    Games     Minecraft
***3   ** 921783    Books     Paula Hawki
***4   ** 216494    Games     Civilizatio
***5   ** 294735    Films     The Grand B
***6   **
***7   ** 813387    Music     Dire Strait
***8   ** 692318    Music     George Ezra
***9   ** 567139    Books     Fflur Dafyd

OVERFLOW AREA

672297    Music     Dvorak: New World Sym
278445    Films     James Bond: Spectre
671120    Books     Kant: Critique of Pur
783910    Books     Albert Camus:  La Pes
```

Find record

Product code: 672297

Category: Music

Title / description: Dvorak: New World Symphony, No. 9 in E minor

Close the program windows and return to the NetBeans editing screen. You may have noticed that it is possible to update the ***category*** and ***title / description*** fields of the records, but not the ***product code***. This was a deliberate design decision, to avoid upsetting the file structure. If the user enters an incorrect product code, they must delete the complete record and re-enter it correctly. We will work on the delete function next.

Use the ***Design*** tab to move to the form layout view, then double click the '***Delete record***' button to create a method. We will add a line of code to the button click method to call ***deleteRecord( )***, then insert the ***deleteRecord( )*** method immediately below the button click method.

We begin the ***delete( )*** method by asking the user to confirm that they wish to delete the record. Please note that the line beginning

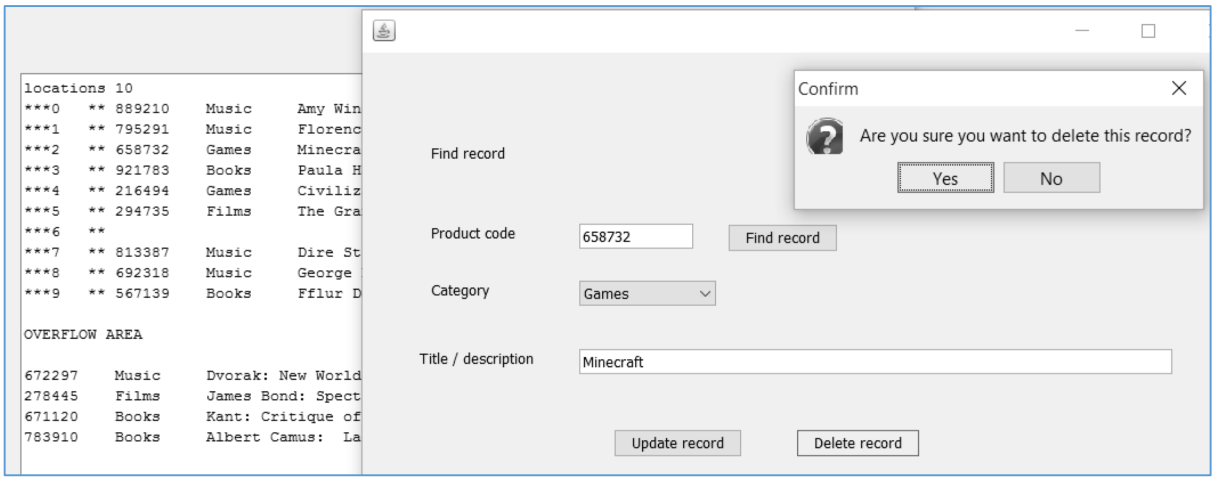
int response = JOptionPane.showConfirmDialog(…

should be entered as a single line of code with no line breaks.

The program checks whether the record to be deleted is in a main file location. If so, the record will be replaced at that position by a blank record.

```
    private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
        deleteRecord();
    }
    private void deleteRecord()
    {
        String s="";
        s=String.format("%-90s", s);
        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)
        {
            if (overflowLocation<0)
            {
                try(RandomAccessFile file = new RandomAccessFile(data.filename, "rw"))
                {
                    int position=100*n + 25;
                    file.seek(position);
                    file.write(s.getBytes());
                    file.close();
                    JOptionPane.showMessageDialog(findRecord.this, "Record deleted");
                }
                catch(IOException e)
                {
                    JOptionPane.showMessageDialog(findRecord.this, "File error");
                }
            }
            txtProductCode.setText("");
            txtDescription.setText("");
        }
    }
```

Run the program. Click the '***Refresh file display***' button to show list the records in the files. Click the '***Find record***' menu option, then select one of the records stored in the main file. Click the '***Delete record***' button, then confirm to delete.



```
locations 10
***0   ** 889210    Music     Amy Win
***1   ** 795291    Music     Florenc
***2   ** 658732    Games     Minecra
***3   ** 921783    Books     Paula H
***4   ** 216494    Games     Civiliz
***5   ** 294735    Films     The Gra
***6   **
***7   ** 813387    Music     Dire St
***8   ** 692318    Music     George
***9   ** 567139    Books     Fflur D

OVERFLOW AREA

672297     Music     Dvorak: New World
278445     Films     James Bond: Spect
671120     Books     Kant: Critique of
783910     Books     Albert Camus:  La
```

Confirm

Are you sure you want to delete this record?

Yes No

Find record

Product code: 658732 Find record

Category: Games

Title / description: Minecraft

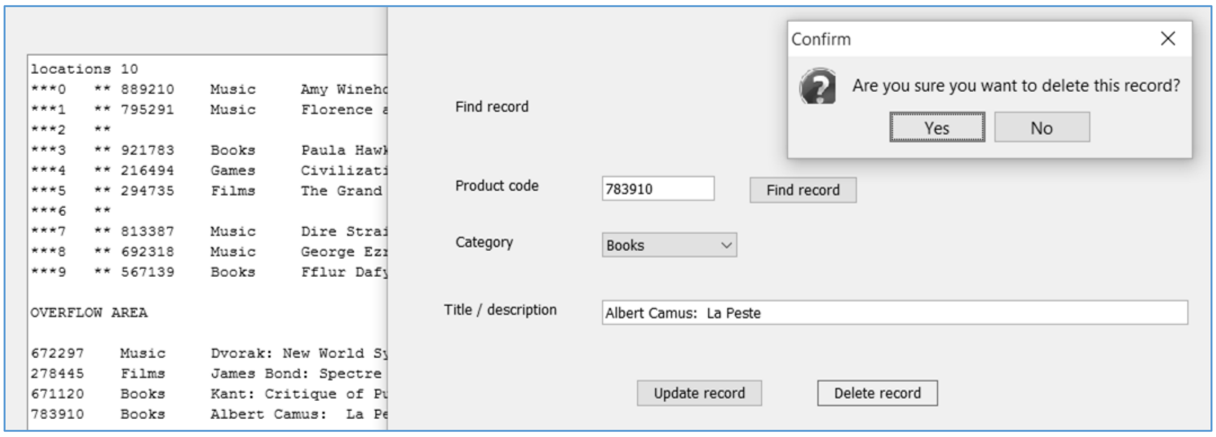
Update record Delete record

Refresh the listing and check that the record has been deleted correctly.

Return to the NetBeans editing screen, and locate the ***delete( )*** method. Add the remaining code needed to delete a record from the overflow area.

```
            catch(IOException e)
            {
                JOptionPane.showMessageDialog(findRecord.this, "File error");
            }
        }
        else
        {
            try(RandomAccessFile file = new RandomAccessFile(data.overflow, "rw"))
            {
                int position=90* overflowLocation;
                file.seek(position);
                file.write(s.getBytes());
                file.close();
                JOptionPane.showMessageDialog(findRecord.this, "Record deleted");
            }
            catch(IOException e)
            {
                JOptionPane.showMessageDialog(findRecord.this, "File error");
            }
        }
        txtProductCode.setText("");
        txtDescription.setText("");
```

Run the program. Repeat the test, this time deleting a record from the ***overflow area***.



```
locations 10
***0   ** 889210    Music     Amy Winehc
***1   ** 795291    Music     Florence &
***2   **
***3   ** 921783    Books     Paula Hawk
***4   ** 216494    Games     Civilizati
***5   ** 294735    Films     The Grand
***6   **
***7   ** 813387    Music     Dire Strai
***8   ** 692318    Music     George Ezi
***9   ** 567139    Books     Fflur Dafy

OVERFLOW AREA

672297     Music     Dvorak: New World Sy
278445     Films     James Bond: Spectre
671120     Books     Kant: Critique of Pu
783910     Books     Albert Camus:  La Pe
```

Confirm

Are you sure you want to delete this record?

Yes No

Find record

Product code: 783910 Find record

Category: Books

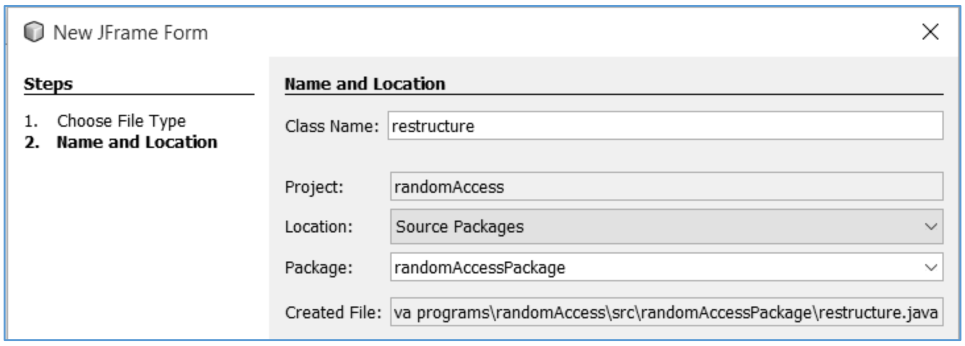
Title / description: Albert Camus: La Peste

Update record Delete record

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

The program that we have created is working correctly, but you might have identified a serious problem with the design. If many more records are added, the main random access storage area will be filled and further records will be stored as overflow. The overflow area uses a simple unsorted file, so has to be searched by the linear search method. For records on disc, this will be very slow. The solution is to restructure the file with a larger number of file locations, then reallocate the existing records in the main file and overflow area using a different hash function. We will do this now…

Go to the Projects window at the top left of the editing screen and right-click on the ***randomAccessPackage*** folder. Select ***New*** / ***JFrame Form***. Give the Class Name as ***restructure***, and leave the Package name as ***randomAccessPackage***.



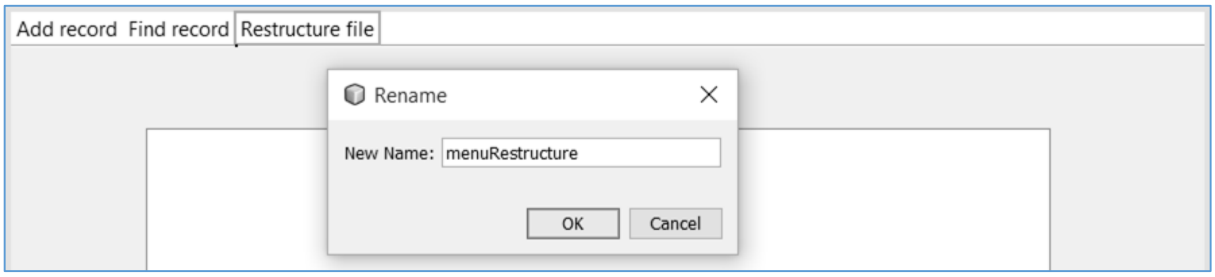
Class Name: restructure
Project: randomAccess
Location: Source Packages
Package: randomAccessPackage
Created File: va programs\randomAccess\src\randomAccessPackage\restructure.java

Click the Finish button to return to the editing screen. The new ***restructure*** form should appear.

- 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***'.

We will now link the new page into the menu system. Use the tab above the editing window to select the ***randomAccess.java*** page. Click the ***Design*** tab to move to the form layout view.

Select a ***Menu*** component from the palette, then drag and drop this onto the Menu Bar. Right-click to change the text caption to '***Restructure file***'. Rename the Menu as ***menuRestructure***.



New Name: menuRestructure

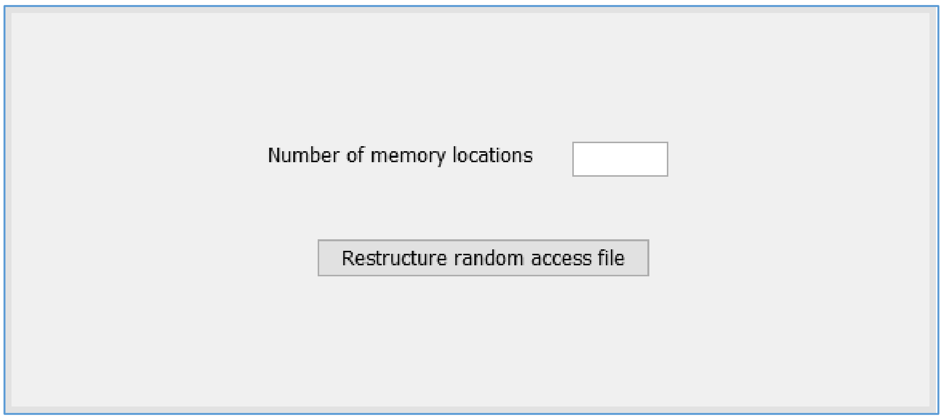
With the '***Restructure file***' menu selected, go to the Properties window and click the ***Events*** tab. Locate the ***mouseClicked*** event, and accept ***menuRestructureMouseClicked*** from the drop down list. Add a line of code to the ***mouseClicked( )*** method to open the ***restructure*** form.

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

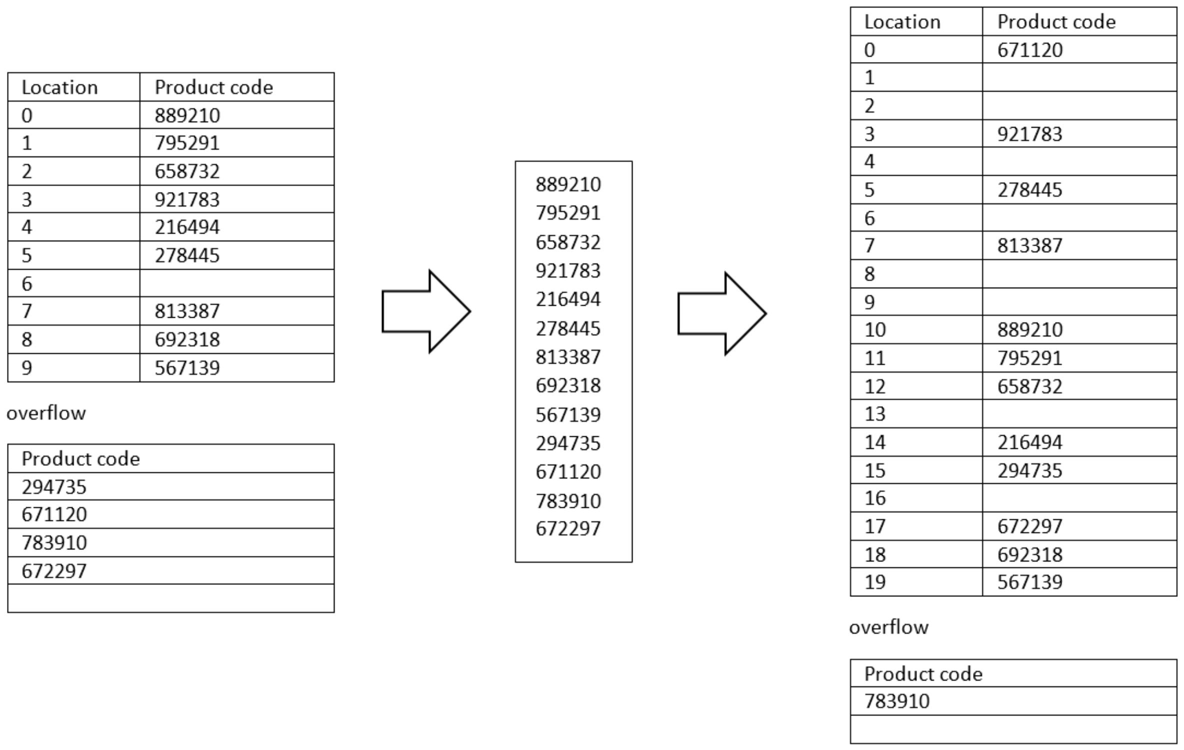
Run the program. Select the '***Restructure file***' menu option and check that the new window opens correctly. It should be possible to close this window by clicking the cross icon without closing the main program.

Close the program and return to the NetBeans editing screen. Click the tab above the editing window to go to the ***restructure.java*** page. Add components to the form:

- A label with the caption '***Number of memory locations***'. Place a text field alongside and rename this as ***txtLocationsWanted***.
- A button with the caption '***Restructure random access file***'. Rename the button as ***btnRestructure***.



Our object will now be to increase the number of memory locations in the main random access storage area, so that the number of overflow records is reduced and the access times are improved. There are currently 10 random access locations. If, for example, the number was increased to 20 then some records in the overflow area could now move to the main file:



Location	Product code
0	889210
1	795291
2	658732
3	921783
4	216494
5	278445
6	
7	813387
8	692318
9	567139

overflow

Product code
294735
671120
783910
672297

```
889210
795291
658732
921783
216494
278445
813387
692318
567139
294735
671120
783910
672297
```

Location	Product code
0	671120
1	
2	
3	921783
4	
5	278445
6	
7	813387
8	
9	
10	889210
11	795291
12	658732
13	
14	216494
15	294735
16	
17	672297
18	692318
19	567139

overflow

Product code
783910

Restructuring the file involves several steps:

- We will copy all the original records from the random access file and overflow area into a temporary file.
- A new empty random access file will be created with the required number of storage locations.
- The records in the temporary file will then be reallocated to the larger random access file using a new hash function, for example:

  <product code> MOD 20

  Any collisions will be handled by moving the records to a new overflow area.

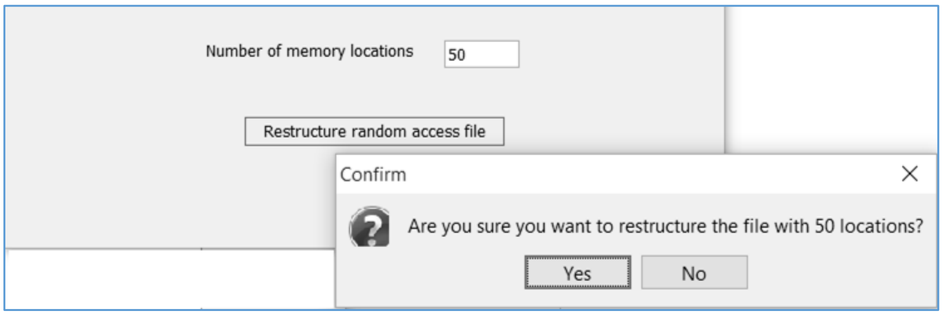
Begin the programming by double clicking the '***Restructure random access file***' button to create a method. Add a line of code to call a ***restructure( )*** method, then add this method immediately underneath. Please note that the line beginning '***int response = JOptionPane.showConfirmDialog*** ' should be entered as a single line of code with no line breaks.
The ***restructure( )*** method begins by collecting the number of storage locations required, then asks the user to confirm that they wish to restructure the file. We then create an empty temporary file into which the existing records can be copied.

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

   restructure();

}
private void restructure()
{
   int position;
   int overflowLocations;

   String s=txtLocationsWanted.getText();
   int locationsWanted=Integer.parseInt(s);
   int response = JOptionPane.showConfirmDialog(null,
             "Are you sure you want to restructure the file with "+
             locationsWanted+" locations?", "Confirm", JOptionPane.YES_NO_OPTION,
             JOptionPane.QUESTION_MESSAGE);
   if (response == JOptionPane.YES_OPTION)
   {
       try
       {
          RandomAccessFile mainFile = new RandomAccessFile(data.filename, "r");
          RandomAccessFile overflowFile = new RandomAccessFile(data.overflow, "r");
          RandomAccessFile tempFile = new RandomAccessFile("temp.dat", "rw");
          tempFile.setLength(0);
       }
       catch(IOException e)
       {
           JOptionPane.showMessageDialog(restructure.this, "File error");
       }
   }
}
```

Scroll up to the top of the program listing and add the Java modules which will be needed for file handling and to create message boxes, as shown below.

```
        package randomAccessPackage;

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

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

Run the program. Select the '***Restructure file***' menu option, then enter a number of memory locations for the new file. Click the button and check that a confirm message is displayed correctly.



Number of memory locations 50

Restructure random access file

Confirm

Are you sure you want to restructure the file with 50 locations?

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

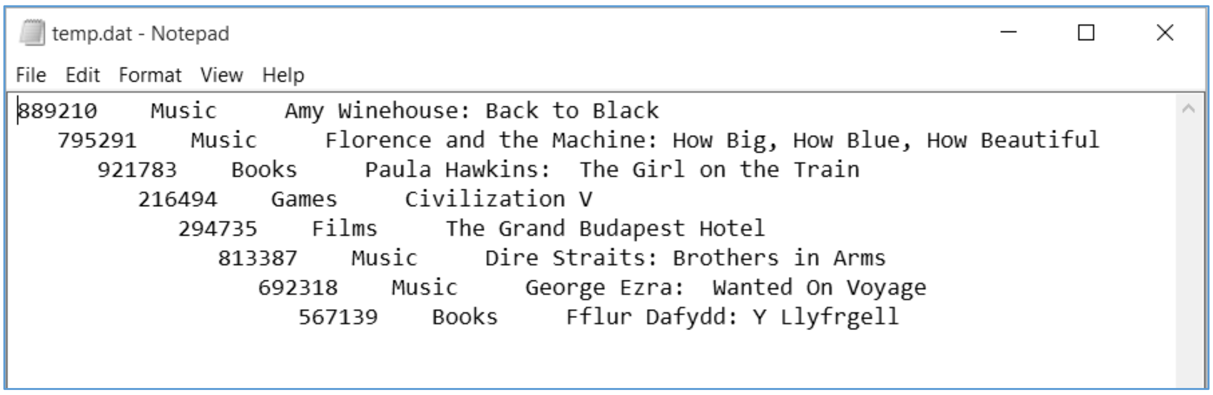
Add the lines of code below which use a loop to obtain each of the product records from the current main file. The records are then copied into the temporary file.

```
    try
    {
        RandomAccessFile mainFile = new RandomAccessFile(data.filename, "r");
        RandomAccessFile overflowFile = new RandomAccessFile(data.overflow, "r");
        RandomAccessFile tempFile = new RandomAccessFile("temp.dat", "rw");
        tempFile.setLength(0);

        for (int i=0; i<data.fileLocations; i++)
        {
            position=100*i+25;
            mainFile.seek(position);
            byte[] bytes = new byte[90];
            mainFile.read(bytes);
            String record=new String(bytes);
            s=record.trim();
            if (s.length() >0)
            {
                tempFile.write(bytes);
            }
        }
        mainFile.close();

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

Run the program. Use the '***Restructure file***' menu option, then enter a number of storage locations for the new file. Click the button and confirm to continue with the restructuring.

Use Windows Explorer to locate the ***temp.dat*** file in the ***randomAccess*** project folder. Open the file using a text editing application such as Notepad.

Check that the records currently in the **main file** are listed.



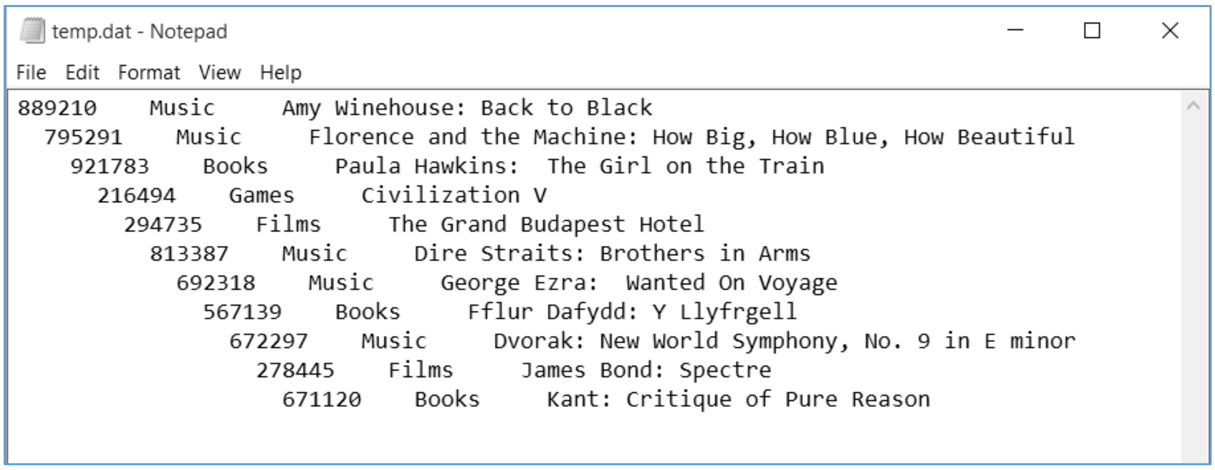
```
889210    Music     Amy Winehouse: Back to Black
   795291    Music     Florence and the Machine: How Big, How Blue, How Beautiful
      921783    Books     Paula Hawkins:  The Girl on the Train
         216494    Games     Civilization V
            294735    Films     The Grand Budapest Hotel
               813387    Music     Dire Straits: Brothers in Arms
                  692318    Music     George Ezra:  Wanted On Voyage
                     567139    Books     Fflur Dafydd: Y Llyfrgell
```

Close the program windows and return to the NetBeans editing screen. Add lines of code to the ***restructure( )*** method which will also add records from the ***overflow area*** to the ***temp.dat*** file.

```
        for (int i=0; i<data.fileLocations; i++)
        {
            position=100*i+25;
            mainFile.seek(position);
            byte[] bytes = new byte[90];
            mainFile.read(bytes);
            String record=new String(bytes);
            s=record.trim();
            if (s.length() >0)
            {
                tempFile.write(bytes);
            }
        }
        mainFile.close();

        overflowLocations=(int) overflowFile.length()/90;
        for (int i=0; i<overflowLocations; i++)
        {
            position=90*i;
            overflowFile.seek(position);
            byte[] bytes = new byte[90];
            overflowFile.read(bytes);
            String record=new String(bytes);
            s=record.trim();
            if (s.length() >0)
            {
                tempFile.write(bytes);
            }
        }
        overflowFile.close();

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

Run the program. As before, use the '***Restructure file***' menu option then enter a number of storage locations for the new file. Click the button and confirm to continue with the restructuring.

Use Windows Explorer to locate the ***temp.dat*** file in the ***randomAccess*** project folder. Open the file using a text editing application. Check that all records from the overflow area have now been added to the file.



```
889210     Music     Amy Winehouse: Back to Black
  795291     Music     Florence and the Machine: How Big, How Blue, How Beautiful
    921783     Books     Paula Hawkins:  The Girl on the Train
      216494     Games     Civilization V
        294735     Films     The Grand Budapest Hotel
          813387     Music     Dire Straits: Brothers in Arms
            692318     Music     George Ezra:  Wanted On Voyage
              567139     Books     Fflur Dafydd: Y Llyfrgell
                672297     Music     Dvorak: New World Symphony, No. 9 in E minor
                  278445     Films     James Bond: Spectre
                    671120     Books     Kant: Critique of Pure Reason
```

Close the program windows and return to the NetBeans editing screen. We will now complete the remaining stages of the strategy to restructure the file.

Begin by adding lines of code to the ***restructure( )*** method which will:

- Update the number of storage locations in the ***data class*** global variable, so that this value will be available to other parts of the program.
- Call a ***createMainFile( )*** method which will rebuild an empty random access file with the required number of storage locations.
- Create a new empty overflow file.

```
            for (int i=0; i<overflowLocations; i++)
            {
                position=90*i;
                overflowFile.seek(position);
                byte[] bytes = new byte[90];
                overflowFile.read(bytes);
                String record=new String(bytes);
                s=record.trim();
                if (s.length() >0)
                {
                    tempFile.write(bytes);
                }
            }
            overflowFile.close();

            data.fileLocations=locationsWanted;
            createMainFile();
            mainFile=new RandomAccessFile(data.filename, "rw");
            overflowFile = new RandomAccessFile(data.overflow, "rw");
            overflowFile.setLength(0);

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

Add the ***createMainFile( )*** method immediately underneath the ***restructure( )*** method.

```
private void createMainFile()
{
    int position;
    try (RandomAccessFile file = new RandomAccessFile(data.filename, "rw"))
    {
        file.setLength(0);
        String s = "locations "+data.fileLocations;
        s=String.format("%-15s", s);
        file.write(s.getBytes());
        for (int i=0; i<data.fileLocations; i++)
        {
           String locationNumber=String.format("%-4s", i);
           s = "***"+locationNumber+"** ";
           s=String.format("%-100s", s);
           position=i*100+15;
           file.seek(position);
           file.write(s.getBytes());
        }
        file.close();
    }
    catch(IOException e)
    {
          JOptionPane.showMessageDialog(restructure.this, "File error");
    }
}
```

Return to the ***restructure( )*** method and add lines of code which will access each record from the ***temp.dat*** file, ready to allocate it to a location in the new random access file.

```
        data.fileLocations=locationsWanted;
        createMainFile();
        mainFile=new RandomAccessFile(data.filename, "rw");
        overflowFile = new RandomAccessFile(data.overflow, "rw");
        overflowFile.setLength(0);

        int tempLocations=(int) tempFile.length()/90;
        for (int i=0; i<tempLocations; i++)
        {
            position=90*i;
            tempFile.seek(position);
            byte[] bytes = new byte[90];
            tempFile.read(bytes);
            String oldRecord=new String(bytes);
            s=oldRecord.trim();
            if (s.length() >0)
            {
                String productCode=oldRecord.substring(0,6).trim();
                int n=getHashValue(productCode);
            }
        }
        tempFile.close();
        JOptionPane.showMessageDialog(restructure.this, "File restructured");

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

A function will be needed to calculate the hash value for each record, using the new number of storage locations. Add the method ***getHashValue( )*** immediately below the ***restructure( )*** method.

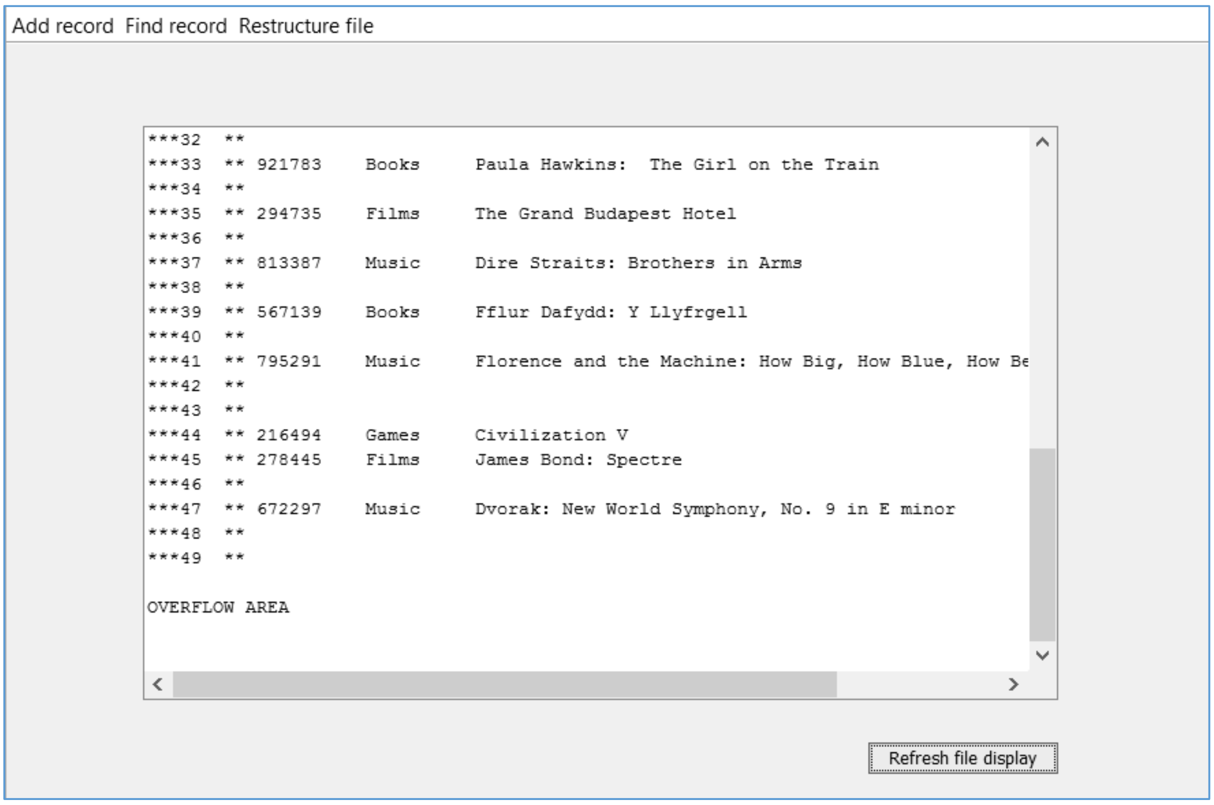
```
private int getHashValue(String productCode)
{
    int hashValue=-1;
    try
    {
        long code=Long.parseLong(productCode);
        hashValue = (int) (code % data.fileLocations);
    }
    catch (NumberFormatException e)
    {

    }
    return hashValue;
}
```

Return to the ***restructure( )*** method. Add lines of code to store the records in the new main file, or in the new overflow area if a collision occurs.

```
        for (int i=0; i<tempLocations; i++)
        {
            position=90*i;
            tempFile.seek(position);
            byte[] bytes = new byte[90];
            tempFile.read(bytes);
            String oldRecord=new String(bytes);
            s=oldRecord.trim();
            if (s.length() >0)
            {
                String productCode=oldRecord.substring(0,6).trim();
                int n=getHashValue(productCode);

                position=n*100 + 25;
                mainFile.seek(position);
                bytes = new byte[90];
                mainFile.read(bytes);
                String newFileEntry=new String(bytes);
                String productCodeFound=newFileEntry.substring(0,6);
                productCodeFound=productCodeFound.trim();
                if (productCodeFound.length()<1)
                {
                    position=100*n + 25;
                    mainFile.seek(position);
                    mainFile.write(oldRecord.getBytes());
                }
                else
                {
                    position=(int) overflowFile.length();
                    mainFile.seek(position);
                    overflowFile.write(oldRecord.getBytes());
                }

            }
        }
        tempFile.close();
        JOptionPane.showMessageDialog(restructure.this, "File restructured");
```

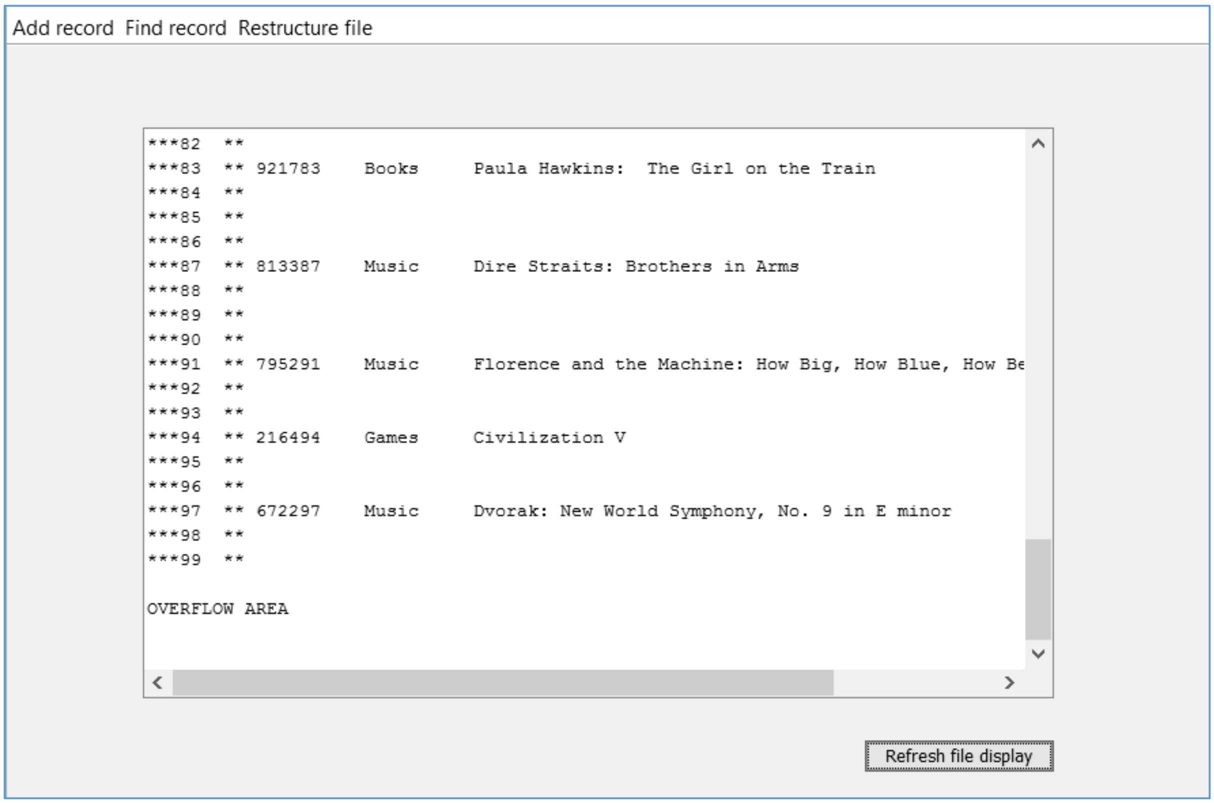
We have now completed the restructuring. Run the program. As before, use the '***Restructure file***' menu option then enter ***50*** as the number of storage locations for the new file. Click the button and confirm to continue with the restructuring.

Return to the main program page and refresh to file display. Check that a random access file with the ***50 storage locations*** has been created, and that the records have been correctly allocated.



```
***32  **
***33  ** 921783    Books     Paula Hawkins:  The Girl on the Train
***34  **
***35  ** 294735    Films     The Grand Budapest Hotel
***36  **
***37  ** 813387    Music     Dire Straits: Brothers in Arms
***38  **
***39  ** 567139    Books     Fflur Dafydd: Y Llyfrgell
***40  **
***41  ** 795291    Music     Florence and the Machine: How Big, How Blue, How Be
***42  **
***43  **
***44  ** 216494    Games     Civilization V
***45  ** 278445    Films     James Bond: Spectre
***46  **
***47  ** 672297    Music     Dvorak: New World Symphony, No. 9 in E minor
***48  **
***49  **

OVERFLOW AREA
```

Return to the '***Restructure file***' window, and now select ***100 storage locations***. Check that the file is again correctly restructured.



```
***82  **
***83  ** 921783    Books     Paula Hawkins:  The Girl on the Train
***84  **
***85  **
***86  **
***87  ** 813387    Music     Dire Straits: Brothers in Arms
***88  **
***89  **
***90  **
***91  ** 795291    Music     Florence and the Machine: How Big, How Blue, How Be
***92  **
***93  **
***94  ** 216494    Games     Civilization V
***95  **
***96  **
***97  ** 672297    Music     Dvorak: New World Symphony, No. 9 in E minor
***98  **
***99  **

OVERFLOW AREA
```