```java
import java.io.Serializable; //needed for Serializable interface
/**this class creates the CustomerFile objects which will store the customer information for each
 * Respective customer. There are "get" and "set" methods to initialise values to each respective
 * Instance variable and the constructor will use the methods to initialise the variables
 *
 * @author Luke Geeson
 * @version 1.00
 * @date 20/11/12
 * @school Norton Knatchbull School
 */
public class CustomerFile implements Serializable //can be serialized and stored in a Serializable file
{
    /**
     * This is the constructor, it will initialise values to the instance variables that create the
     * Customer record using the get and set methods of this class. It does not return anything
     * So it is void but it will output a String ensuring the user that a record has been created
     *
     * @param fName         this is the forename string which will be assigned to forename
     * @param sName         this is the surname string which will be assigned to surname
     * @param pNum          this is the phone number which will be assigned to phoneNumber
     * @param eAddress      this is the email address which will be assigned to eAddress
     * @param hAddress      this is the home address which will be assigned to homeAddress
     * @param pCode         this is the post code which will be assigned to postCode
     */
    public CustomerFile(String fName, String sName, String pNum, String eAddress, String hAddress, String pCode)
    {
        setForename(fName);
        setSurname(sName);
        setPhoneNumber(pNum);
        setEmailAddress(eAddress);
        setHomeAddress(hAddress);
        setPostCode(pCode);
        System.out.println("Details successfully input");
    }
    /**
     * This is the default constructor
     */
    public CustomerFile(){}
    //-----------------------------------------------------------------------------------------------
    /**
     * This is the method that will initialise the variable "forename" - method is void, it doesn't return
     * anything
     *
     * @param a             this is the string that will be assigned to String forename
     */
    public void setForename(String inputForename)
    {
        this.forename = convertAndPresent(inputForename);
    }
    //-----------------------------------------------------------------------------------------------
    /**
     * This is the method that will return the variable "forename"
     *
     * @return forename     this is the variable that stores the forename
     */
    public String getForename()
    {
        return this.forename;
    }
    //-----------------------------------------------------------------------------------------------
    /**
     * This method initialises the variable "surname" with the input string - method is void, it doesn't
     * return anything
     *
     * @param a             this String will be used to initialise the "surname" variable
     */
    public void setSurname(String inputSurname)
    {
        this.surname = convertAndPresent(inputSurname);
    }
    //-----------------------------------------------------------------------------------------------
    /**
     * This method will return the String variable "surname"
     *
     * @return surname      this is the variable that stores the surname
     */
    public String getSurname()
    {
        return this.surname;
    }
    //------------------------------------------------------------------------------------------------
    /**
     * This method initialises the String variable "phoneNumber" - method is void
     *
     * @param a             this string will be used to initialise the "phoneNumber" variable
     */
    public void setPhoneNumber(String inputPNum)
    {
        this.phoneNumber = inputPNum.trim();
    }
    //------------------------------------------------------------------------------------------------
    /**
     * This method returns the String variable "phoneNumber"
     *
     * @return phoneNumber  the variable that stores the phone number
     */
    public String getPhoneNumber()
    {
        return this.phoneNumber;
    }
    //------------------------------------------------------------------------------------------------
    /**
     * This method initialises the variable "emailAddress" - method is void
     *
     * @param a             the String used to initialise the variable "emailAddress"
     */
    public void setEmailAddress(String inputEAddress)
    {
        inputEAddress = inputEAddress.trim();
        this.emailAddress = inputEAddress;
    }
    //------------------------------------------------------------------------------------------------
    /**
     * This method returns the variable "emailAddress"
```

```
         *
         * @return emailAddress the variable that stores the email address
         */
        public String getEmailAddress()
        {
            return this.emailAddress;
        }
        //--------------------------------------------------------------------------------------------------
        /**
         * This method initialises the variable "homeAddress" - method is void
         *
         * @param a             the string used to initialise the variable "homeAddress"
         */
        public void setHomeAddress(String inputHAddress)
        {
            inputHAddress = inputHAddress.trim();
            this.homeAddress = inputHAddress;
        }
        //--------------------------------------------------------------------------------------------------
        /**
         * This method returns the variable "homeAddress"
         *
         * @return homeAddress  the String used to store the home address
         */
        public String getHomeAddress()
        {
            return this.homeAddress;
        }
        //--------------------------------------------------------------------------------------------------
        /**
         * This method initialises the variable "postCode" - method is void
         *
         * @param a             the String used to initialise the variable "postCode"
         */
        public void setPostCode(String inputPCode)
        {
            inputPCode = inputPCode.trim();
            inputPCode = inputPCode.toUpperCase();
```

```java
        this.postCode = inputPCode;
    }
    //------------------------------------------------------------------------------------------------
    /**
     * This method returns the variable "postCode"
     *
     * @return postCode     the string used to store the post code
     */
    public String getPostCode()
    {
        return this.postCode;
    }
    //------------------------------------------------------------------------------------------------
    /**
     * This method converts Strings input so that the first letter is changed to uppercase to be used with proper nouns
     * such as names and days of the year - in this method it changes the String input (the forename and Surname for each file)
     *
     * @param toConvert     the string which will be converted to a proper noun e.g input: "luke" output: "Luke"
     *
     * @return converted    returns the converted String with an appropriate capital letter
     */
    public static String convertAndPresent(String toConvert)
    {
        toConvert = toConvert.trim();
        if (toConvert.equals(""))
        {
            toConvert = "empty field";
        }
        char firstLetter = toConvert.charAt(0);            //takes the first letter
        String converted = String.valueOf(firstLetter).toUpperCase() + toConvert.substring(1,toConvert.length()); //returns the letter (in upper case)
        return converted;                                  //returns this string
    }
    //------------------------------------------------------------------------------------------------
    /**
     * Method used to display all the details of a CustomerFile - method is void as it is printing and
     * takes no parameters
```

```
         */
        public void displayAllDetails()
        {
            System.out.println("Customer name:\t " + getForename() + " " + getSurname());       //prints customer name
            System.out.println("Phone number:\t " + getPhoneNumber());                          //prints phone number
            System.out.println("Email address\t " + getEmailAddress());                         //prints email address
            System.out.println("Home address:\t " + getHomeAddress());                          //prints home address
            System.out.println("Post code:\t " + getPostCode());                                //prints post code
            System.out.print("\n");                                                             //prints gap between text on screen
        }
        //-------------------------------------------------------------------------------------------------
        //state variables used in this class - private for data protection
        private String forename;                   //declares the forename of this customer as a string
        private String surname;                    //declares the surname of the customer as a string
        private String phoneNumber;                //declares the phone number of the customer as a string
        private String emailAddress;               //declares the email address of the customer as a string
        private String homeAddress;                //declares the home address of the customer as a string
        private String postCode;                   //declares the post code of the customer as a string
    }

                                                   //------new class------\\

    import java.io.Serializable;                   //needed for the Serializable interface
    /**
     * This class creates the nodes used in the singly linked list of customer records. This method contains
     * the "get" and "set" methods for the node data and nextNode variables
     *
     * @author Luke Geeson
     * @version 1.00
     * @date 25/11/12
     * @school Norton Knatchbull School
     */
    public class Node implements Serializable   //used so it can be serialized
    {
        /**
         * This is the constructor; it will initialise the variables of the Node object
         *
         * @param data      this is the CustomerFile object that will make up the data of the node
```

```java
     * @param next      this is the next Object in the list
     */
    public Node(CustomerFile data, Node next)
    {
        nextNode = next;
        nodeData = data;
    }
    public Node(){}
    //-------------------------------------------------------------------------------------------------
    /**
     * This method initialises the variable "nodeData" with a CustomerFile object as the data - method is void
     *
     * @param input     this is the Object object which will be assigned to the Object object "nodeData"
     */
    public void setData(CustomerFile input)
    {
        this.nodeData = input;
    }
    //-------------------------------------------------------------------------------------------------
    /**
     * This method returns the CustomerFile object data assigned to "nodeData"
     *
     * @return nodeData this is the CustomerFile object which contains the data for a customer file
     */
    public CustomerFile getData()
    {
        return this.nodeData;
    }
    //-------------------------------------------------------------------------------------------------
    /**
     * This method initialises the Node object "nextNode" with the Node input - method is void
     *
     * @param inputNext the Node which will initialise the variable "nextNode"
     */
    public void setNext(Node inputNext)
    {
        this.nextNode = inputNext;
    }
    //--------------------------------------------------------------------------------------------------
    /**
     * This method returns the Node stored in the variable "nextNode" which is the next node in the linked list
     *
     * @return nextNode the Node which contains the next item in the linked list
     */
    public Node getNext()
    {
        return this.nextNode;
    }
    //--------------------------------------------------------------------------------------------------
    //state variables used in this class - private for data protection
    private CustomerFile nodeData;          //contains the customer record of this node
    private Node nextNode;                  //contains a reference to the next node
}

                                                    //------new class------\\

import java.io.Serializable;                //needed for the Serializable interface
/**
 * This is the class that is used to make a singly linked list with Node objects as the elements. There
 * are methods to add, remove, modify, search, find the size, determine if it is empty, sort, merge sort
 * set/get the head node (add first), set/get the last node (add last), print items in ascending/descending
 * order and a method to convert the linked list into a node array
 *
 * @author Luke Geeson
 * @version 1.00
 * @date 29/11/12
 * @school Norton Knatchbull School
 */
public class SinglyLinkedList implements Serializable
{
    /**
     * this constructor is used when a new linked list is created and you want to pass the first node
     * of the list to it. If there is only one Node passed then pass null as the nextNode parameter
     *
     * @param firstNode         the first node in the linked list, set as the head of the list
     * @param nextNode          the next node after the first, pass as null if there are no other nodes
```

```
         */
        public SinglyLinkedList (Node firstNode, Node nextNode)
        {
            setHead(firstNode);
            firstNode.setNext(nextNode);
        }
        /**
         * This is the default constructor
         */
        public SinglyLinkedList()
        {
            this.head = null;
        }
        //-------------------------------------------------------------------------------------------------
        /**
         * returns a Boolean value dependent on whether the Linked List is empty - no parameters
         *
         * @return empty             will determine whether it is empty - 'true' = the list is empty
         */
        public boolean isEmpty()
        {
            boolean empty;
            if (this.head == null)
            {
                empty = true;                    //if there is not a head node i.e. the list is empty
            }
            else
            {
                empty = false;                   //if there is a head node i.e. the list is full
            }
            return empty;
        }
        //-------------------------------------------------------------------------------------------------
        /**
         * returns the size of the list
         *
         * @param input              calculates the size of the list with this variable - pass head node
         *
         * @return int               the size of the list (null if empty)
         */
        public static int length(Node input)
        {
            if (input == null)
            {
                return 0;                                  //if the node is empty, if the head node and the list is empty
            }
            else
            {
                return 1 + length(input.getNext());        //recursive length check
            }
        }
        //----------------------------------------------------------------------------------------------------
        /**
         * Inserts a Node in the list, in order according to the Surname is void as it does not return anything
         *
         * @param newData           the new Node which contains the new data to be inserted
         * @param trailNode         the node which the new data will compare with
         */
        public void addRecord(Node newData, Node trailNode)
        {
            if(trailNode == null)                          //if the trailNode is null
            {
                if (this.head == null)                     //if the linked list has no head i.e. it is empty
                {
                    this.setHead(newData);                 //sets the new data as the head of the list
                }
                else                                       //if the list is at its end
                {
                    trailNode = newData;                   //sets the newData as the last item
                }
                newData.setNext(null);                     //sets the next item as null if the node is the first or last item
            }
            else if (trailNode == getHead() && newData.getData().getSurname().compareTo(trailNode.getData().getSurname())<0)
            {                                              //if the trail node = head node and the new data comes before it
                newData.setNext(getHead());                //sets the newdata.next to the current head
                setHead(newData);                          //sets the newdata as the head of the list
        }
        else if (trailNode.getNext() == null)               //if the trail.next is empty
        {
            trailNode.setNext(newData);                     //set the next item in the list
            newData.setNext(null);                          //set the next item in the list as null
        }
        else if (newData.getData().getSurname().equalsIgnoreCase(trailNode.getData().getSurname()))
        {                                                   //if the surnames are identical - store new one after old
            newData.setNext(trailNode.getNext());           //set new data.next to trail node.next
            trailNode.setNext(newData);                     //set trailnode.next as new data
        }
        else if(newData.getData().getSurname().compareTo(trailNode.getNext().getData().getSurname())<0)
        {                                                   //if the new item comes before the trail.next
            if (newData.getData().getSurname().compareTo(trailNode.getData().getSurname())>0)
            {                                               //if the new item comes after the trail node
                newData.setNext(trailNode.getNext());       //sets the new data as the node before the item after trail node
                trailNode.setNext(newData);                 //sets the new data as the item after the trail node
            }
        }
        else
        {                                                   //if the new data is not within the confines of the 2 nodes
            addRecord(newData,trailNode.getNext());         //recursive traversal through the list to the correct point
        }
    }
    //------------------------------------------------------------------------------------------------
    /**
     * This method searches for the first occurrence of the Node with an identical key String to "search"
     *
     * @param compNode          the node which will be compared with the search string
     * @param search            the string which that is sought after
     *
     * @return compNode or null this will return the first occurrence of the Node containing the search String null if no such
node is found
     */
    public Node searchList(Node compNode, String search)
    {
        if (compNode == null)
        {
```

```java
            return null;                                //if the list is empty
        }
        else if (compNode.getData().getSurname().equalsIgnoreCase(search))
        {
            return compNode;                            //if the node has the search string
        }
        else
        {
            return searchList(compNode.getNext(),search);   //recursive traversal through the list
        }
    }
    //-------------------------------------------------------------------------------------------------
    /**
     * This method will take a Node from the parameter and remove the first occurrence of it in the singly
     * Linked List. It is void  and takes 2 parameters:
     *
     * @param toRemove          the node which needs to be removed
     * @param compNode          the node with which the 'toRemove' node will be compared - allows recursive traversal
     */
    public void removeNode(Node toRemove, Node compNode)
    {
        if (isEmpty() == true)
        {                                               //if the list is empty
            System.out.println("The list is empty - cannot remove this item as it does not exist");
        }
        else if (toRemove == getHead())
        {                                               //if the head of the list is the node to remove
            if (getHead() != null && getHead().getNext() == null)
            {                                           //if the head is not null but the next node is
                this.head = null;                       //removes the node from the list
            }
            else if (getHead().getNext() != null)
            {                                           //if the next item is not null and the head is to be removed
                toRemove = getHead();                   //marks the head to be removed
                this.head = getHead().getNext();        //the head is set as the next node
                toRemove = null;                        //the old head is deleted
            }
        }
        else if (compNode.getNext() == toRemove && compNode.getNext() == getLast(getHead()))
        {                                                   //if the node to remove is at the end of the list
            compNode.setNext(null);                         //removes the last item in the list
        }
        else if (toRemove == compNode.getNext() )
        {                                                   //if the next item in the list is the item to remove
            toRemove = compNode.getNext();                  //marks the next item to be removed
            Node remPrev = compNode;                        //gets the previous node for reallocation
            Node remAfter = toRemove.getNext();             //gets the next node for reallocation
            remPrev.setNext(remAfter);                      //reallocates other nodes
            toRemove = null;                                //removes old node
        }
        else
        {                                                   //recursive traversal through list until statement is satisfied
            removeNode(toRemove, compNode.getNext());
        }
    }
    //-----------------------------------------------------------------------------------------------
    /**
     * This method will take a node and change the specific details of it based on the data supplied
     * NOTE: this method alters DATA of the node - use get/set data methods to alter the node itself
     *
     * @param nodeToChange      the node which will be altered
     * @param changeDecision    the decision of which variable is to be altered
     * @param changeInput       the new information that will replace the old
     *
     * @return nodeToChange     the old node is return with the changes
     */
    public static Node changeNode(Node nodeToChange, String changeDecision, String changeInput)
    {
        if (changeDecision.equalsIgnoreCase("FORENAME") || changeDecision.equalsIgnoreCase("FIRSTNAME") ||
changeDecision.equalsIgnoreCase("FIRST NAME")|| changeDecision.equalsIgnoreCase("FIRST") ||
changeDecision.equalsIgnoreCase("F"))
        {                                                   //if the forename is to be changed
            nodeToChange.getData().setForename(changeInput);//change the forename of the node
        }
        else if (changeDecision.equalsIgnoreCase("SURNAME") || changeDecision.equalsIgnoreCase("SECONDNAME") ||
changeDecision.equalsIgnoreCase("SECOND NAME") || changeDecision.equalsIgnoreCase("S"))
```

```
        {                                                   //if the surname is to be changed
            nodeToChange.getData().setSurname(changeInput); //change the surname
        }
        else if (changeDecision.equalsIgnoreCase("PHONENUMBER") || changeDecision.equalsIgnoreCase("PHONE NUMBER") ||
changeDecision.equalsIgnoreCase("PHONE"))
        {                                                   //if the phone number is to be changed
            nodeToChange.getData().setPhoneNumber(changeInput); //change it
        }
        else if(changeDecision.equalsIgnoreCase("EMAILADDRESS") || changeDecision.equalsIgnoreCase("EMAIL ADDRESS") ||
changeDecision.equalsIgnoreCase("EMAIL") || changeDecision.equalsIgnoreCase("E"))
        {                                                   //if the email address is to be changed
            nodeToChange.getData().setEmailAddress(changeInput);//change it
        }
        else if (changeDecision.equalsIgnoreCase("HOMEADDRESS") || changeDecision.equalsIgnoreCase("HOME ADDRESS") ||
changeDecision.equalsIgnoreCase("HOME") || changeDecision.equalsIgnoreCase("H"))
        {                                                   //if the home address is to be changed
            nodeToChange.getData().setHomeAddress(changeInput);//change it
        }
        else if (changeDecision.equalsIgnoreCase("POSTCODE") || changeDecision.equalsIgnoreCase ("POST CODE"))
        {                                                   //if the postcode is to be changed
            nodeToChange.getData().setPostCode(changeInput);//change it
        }
        return nodeToChange;                                //return the modified node
    }
    //-------------------------------------------------------------------------------------------------
    /**
     * This method performs a bubble sort on the linked list and sorts the data in ascending order
     * performs a sort on the list associated and returns nothing
     */
    public void sortList()
    {
        if (isEmpty())                                      //if the list is empty
        {
            System.out.println("cannot sort list - list is empty\n");
        }
        else if (getHead().getNext() == null)               //if the list has one item only
        {
            System.out.println("List sorted\n");
            }
            else
            {
                Node current = getHead();                       //starts sort from the start of the  list
                boolean swapDone = true;                        //used to iterate the loop
                while (swapDone == true)
                {
                    swapDone = false;                           //used to escape the loop
                    while (current != null)
                    {                                           //while current is not equal to the last item
                        if (current.getNext() != null &&
current.getData().getSurname().compareTo(current.getNext().getData().getSurname()) >0)
                        {                                       //if the next item comes before the current
                            CustomerFile tempDat = current.getData();
                            current.setData(current.getNext().getData());
                            current.getNext().setData(tempDat); //swap the data
                            swapDone = true;                    //used to continue the loop
                        }
                        current = current.getNext();            //traversal through the loop
                    }
                    current = getHead();                        //continue at start of the loop
                }
                System.out.println("List sorted\n");
            }
        }
        //-----------------------------------------------------------------------------------------------
        /**
         * This method will take two singly linked list objects as parameters and merge them - it has 4
         * possible ways of working: 1.if both are empty, 2.one is full and the other is not, 3. vice versa
         * or 4.both are full.
         *
         * @param lst1              the first list
         * @param lst2              the second list
         *
         * @return newList          returns a new list with the 2 lists merged
         */
        public static SinglyLinkedList mergeLists(SinglyLinkedList lst1, SinglyLinkedList lst2)
        {
```

```
        SinglyLinkedList newList = new SinglyLinkedList();  //new list for merging
        if (lst1 == null && lst2 == null)                   //if both lists are empty
        {
            newList = null;                                 //cannot merge what is not there
            System.out.println("Both lists are empty - cannot merge");
        }
        else if (lst1 == null && lst2 != null)              //if list 1 is empty but the other is not
        {
            newList = lst2;                                 //set the new list as list 2
        }
        else if (lst1 != null && lst2 == null)              //if list 2 is empty but the other is not
        {
            newList = lst1;                                 //set the new list as list 1
        }
        else                                                //if both lists have data
        {
            newList = lst1;                                 //set the new list as list 1 and add all records from the second
            Node current = lst2.getHead();                  //start point for adding to the new list from list 2
            for (int j = 0; j < lst2.length(lst2.getHead()); j++)
            {
                newList.addRecord(current,newList.getHead()); //add record method inserts in order
                current = current.getNext();                //iterative approach to merging the lists - allows traversal
            }
        }
        return newList;                                     //returns the new list
    }
    //-------------------------------------------------------------------------------------------------
    /**
     * prints the linked list in descending order  to the node which is input
     *
     * @param input              prints all items from this point onward in descending order
     */
    public void printReverseList(Node input)
    {
        if (input != null)
        {
            printReverseList(input.getNext());              //recursive progression to the end of the list
            input.getData().displayAllDetails();            //prints the information of the node
            }
        }
        //-----------------------------------------------------------------------------------------------
        /**
         * Prints the entire linked list in ascending order from the node input - will print from any node
         * in the list to the end of the list but to print the whole list - pass the head node
         *
         * @param input             the node from which you print
         */
        public void printList(Node input)
        {
            if(input != null)
            {
                input.getData().displayAllDetails();            //prints all data from the node passed
                printList(input.getNext());                     //recursive print
            }
        }
        //-----------------------------------------------------------------------------------------------
        /**
         * assigns the value input to the "head" variable and sets as the head of the list
         *
         * @param input             a Node which will be set as the head
         */
        public void setHead(Node input)
        {
            input.setNext(this.head);                           //sets the head as the node after the input node
            head = input;                                       //sets the node input as the head node
        }
        //-----------------------------------------------------------------------------------------------
        /**
         * Returns the value assigned to variable "head" effectively setting the head
         * and adding a node to the list
         *
         * @return head             the head of the list
         */
        public Node getHead()
        {
            return this.head;
```

```
    }
    //-----------------------------------------------------------------------------------
    /**
     * adds a node to the end of the list
     *
     * @param input             a node which will be set as the last item in the list
     * @param elementOfList     a node which will be used to cycle through the list and append the input node to the end
     */
    public void appendLast(Node elementOfList,Node input)
    {
        if (elementOfList == null)                          //if the node is empty
        {
            if (isEmpty() == true)                          //if the list is empty
            {
                setHead(input);                             //sets the item as the new head
                input.setNext(null);                        //sets the next as null - it is the only item in the list
            }
            else                                            //if the list is not empty
            {
                elementOfList = input;                      //sets the last item in the list as the input node
                elementOfList.setNext(null);                //creates an end point to the list
            }
        }
        else if (elementOfList.getNext() == null)           //if the next item in the list is null
        {
            elementOfList.setNext(input);                   //set the last item as the new node
            input.setNext(null);                            //the new node is now the last item
        }
        else                                                //recursive traversal through the list if these are not satisfied
        {
            appendLast(elementOfList.getNext(), input);
        }
    }
    //-----------------------------------------------------------------------------------
    /**
     * Returns the value assigned to the last item in the list
     *
     * @param elementOfList     used to cycle through the list and reach the end of the list
```

```
         * @return elementOfList    returned as the last item in the list
         */
        public Node getLast(Node elementOfList)
        {
            if (elementOfList == null)
            {
                return elementOfList;                           //if the linked list is empty - returns null
            }
            else if (elementOfList.getNext() == null)
            {
                return elementOfList;                           //returns the last item in the list
            }
            else
            {
                return getLast(elementOfList.getNext());        //recursive traversal through the list
            }
        }
        //-------------------------------------------------------------------------------------------------
        /**
         * converts the Linked List to a data array so that the data contained in an array will itself be an array
         * - no parameters
         *
         * @return dataArray        an object array that will contain the data within each node of the linked list
         */
        public Node[] toArray()
        {
            int k = SinglyLinkedList.length(this.head);        //gets the size of the list
            Node dataArray[] = new Node[k];                    //initialises a Node array of the size of the list
            Node listNode = getHead();                         //gets the head of the  list
            for (int j = 0; j<k-1;j++)
            {
                dataArray[j] = listNode;                       //initialises each item of the list to an element in the array
                listNode = listNode.getNext();                 //gets the next item in the list
            }
            return dataArray;                                  //returns the array once complete
        }
        //-------------------------------------------------------------------------------------------------
        //state variables - private for data protection
```

```
    private Node head;                                          //the head of the list
}

                                                        //------new class------\\

import java.io.*;                                           //used for file i/o and Scanner
import java.util.*;                                         //used for the Scanner object
import java.io.IOException.*;                               //used to catch exceptions
import java.io.ObjectOutputStream;                          //used to write data to secondary memory using a data stream
import java.io.ObjectInputStream;                           //used to read data from secondary memory using a data stream
/**
 * This is the main class that manages with file I/O and getting commands and information from the user
 * that can be set into fields using the get/set methods. There is a main method that will be used to
 * acquire this.
 *
 * @author Luke Geeson
 * @version 1.20
 * @date 12/12/2012
 * @school Norton Knatchbull School
 */
public class mainClass
{
    /**
     * This is the default constructor
     */
    public mainClass(){}
    //-----------------------------------------------------------------------------------------------
    /**
     * this method acts allows the user to input commands and information to the various methods. This
     * method will interpret the commands and pass control to the other methods of this class which need it
     * void - does not return anything - does not return anything
     */
    public static void main ()
    {
        boolean correctPas = false;                         //Boolean for the password loop
        int attemptNo = 0;                                  //used to allow 3 attempts
        String passwordAttempt = "";                        //string for password attempts
        Scanner kbReader = new Scanner (System.in);         //scanner used throughout method
        while (correctPas != true)                         //password input loop
        {
            if ( attemptNo == 3)                           //if the user has attempted to login 3 times
            {
                passwordAttempt = null;                    //same variable is used to input system code
                while (passwordAttempt == null)            //will loop until system code is input
                {
                    System.out.println("Login failed too many times, please input the SYSTEM PASSCODE");
                    String systemCode = kbReader.nextLine();//system code input
                    passwordAttempt = getPassword(systemCode);//using the system code to get the password
                }
            }
            else
            {
                System.out.println("Input login password - it is case sensitive ");
                passwordAttempt = kbReader.nextLine();
            }
            if (getPasswordWithoutCode().equals("") || getPasswordWithoutCode().equals(" ") || getPasswordWithoutCode() == null)
            {                                              //if the default password is nothing or null
                setPassword("password");
            }
            if (passwordAttempt.equals(getPasswordWithoutCode()))
            {                                              //if the password is correct
                correctPas = true;                         //used to leave the loop
                Date date = new Date();                    //get the date of login
                System.out.println("Login successful - Welcome, the date is " + date.toString() + "\n");
            }
            else
            {                                              //if the password is incorrect
                System.out.println("Incorrect password - please try again " + (2-attemptNo) + " attempts remaining\n");
                if (attemptNo < 3)                         //increments for login attempts
                {
                    attemptNo++;                           //increment the attempts by 1
                }
                correctPas = false;                        //continue the loop
            }
        }
```

```java
        SinglyLinkedList a = loadListFromFile();            //load the list from the file
        boolean done = false;                               //Boolean for the main loop
        while (done != true)                                //loop until the user is finished
        {
            saveListToFile(a); //saves the list after each successive - keeps everything up to date
            System.out.println("What do you want to do? add/search/Print all/get size/sort/clear list/change password or EXIT 
to quit");
            String choice = kbReader.nextLine();            //takes user request
            choice.trim();                                  //trims request for useless space
            if(choice.equalsIgnoreCase("ADD") || choice.equalsIgnoreCase("A"))
            {                                               //interprets user 'add' request
                System.out.println("Input Forename");       //add forename
                String fName = new String (kbReader.nextLine());
                boolean hasSurname = false;                 //loop for surname
                String sName = "";
                while (hasSurname != true)                  //while an empty surname is NOT input
                {
                    System.out.println("Input Surname");    //acquire user input
                    sName = new String (kbReader.nextLine());
                    if (sName.equalsIgnoreCase("") || sName == null )
                    {                                       //if the surname input is empty
                        System.out.println("Invalid input - please insert a surname");
                    }
                    else
                    {                                       //if the surname input is not empty
                        hasSurname = true;
                    }
                }
                System.out.println("Input Phone Number");  //add phone number
                String pNum = new String (kbReader.nextLine());
                System.out.println("Input Email Address"); //add email address
                String eAdd = new String (kbReader.nextLine());
                System.out.println("Input the first line of the Home Address"); //add home address
                String hAdd = new String (kbReader.nextLine());
                System.out.println("Input Post Code");     //add post code
                String pCode = new String (kbReader.nextLine());
                CustomerFile custDat = new CustomerFile(fName,sName,pNum,eAdd,hAdd,pCode); //create customer file for data
                Node custNode = new Node(custDat, null);    //create node to store data
                if (a == null || a.getHead() == null)      //if the list is empty
                {
                    a = new SinglyLinkedList();            //create new list
                }
                a.addRecord(custNode, a.getHead());        //adds the record to the list
                System.out.println("File added successfully\n");
            }
            else if(choice.equalsIgnoreCase("SEARCH") || choice.equalsIgnoreCase("S"))
            {                                              //interprets the users 'search' request
                Node result = null, current = null;        //nodes used for comparison and search results
                boolean foundFile = false;                 //used for while loop
                String search = "";                        //used for surname input for search
                while (foundFile != true)                  //while the record is not found
                {
                    if (result == null)                    //if this is the first search
                    {
                        System.out.println("Input Surname of customer required");
                        search = kbReader.nextLine();
                        search = search.trim();
                    }
                    if (a == null)                         //if the list is empty
                    {
                        System.out.println("Search failed - the list is empty\n");
                        foundFile = true;
                        continue;
                    }
                    else if (current != null)              //is this is a second search
                    {
                        if (current.getNext() == null)
                        { //if the previous search yielded a wrong record and it is the last item (or only item) in the list
                            System.out.println("search failed");
                            foundFile = true;
                            continue;
                        }
                        else
                        {                                  //will continue search from the previous point in the list
                            result = a.searchList(current.getNext(), search);
                        }
```

```
        }
        else                                       //search from the beginning
        {
            result = a.searchList(a.getHead(), search);
        }
        if (result == null || search == "")    //if the search fails OR the search input is empty
        {
            System.out.println("Search failed, please try again");
            foundFile = true;
            break;
        }
        else                                       //if a record is found
        {
            System.out.println("search successful - printing details of file\n");
            result.getData().displayAllDetails();
        }
        System.out.println("Is this the record you need? yes/no");
        String validation = kbReader.nextLine();//request confirmation for found record
        if (validation.equalsIgnoreCase("YES") || validation.equalsIgnoreCase("Y"))
        {                                          //interpret user confirmation for the correct record
            current = result;
            foundFile = true;
            boolean doneEdittingFile = false;  //used for modification or remove loop
            while (doneEdittingFile != true)
            {
                System.out.println("What would you like to do with this record? remove/modify or EXIT");
                String remOrMod = kbReader.nextLine(); //requests action to be performed on record
                if (remOrMod.equalsIgnoreCase("REMOVE") || remOrMod.equalsIgnoreCase("R"))
                {
                    boolean finRemove = false; //used for remove loop
                    while (finRemove != true)
                    {
                        System.out.println("Are you sure that you want to remove this file? yes/no");
                        String confirm = kbReader.nextLine();//confirm that the user wants to remove the file
                        if (confirm.equalsIgnoreCase("YES") || confirm.equalsIgnoreCase("Y"))
                        {                          //if they do
                            a.removeNode(current, a.getHead());
                            System.out.println("Record successfully removed\n");
```

```java
                                        finRemove = true;
                                    }
                                    else if(confirm.equalsIgnoreCase("NO") || confirm.equalsIgnoreCase("N"))
                                    {                       //if they do not
                                        System.out.println("File not removed\n");
                                        result = null;
                                        current = null;
                                        finRemove = true;
                                    }
                                    else
                                    {                       //if the user inputs invalid data
                                        System.out.println("Invalid input - please try again");
                                        finRemove = false;
                                    }
                                }
                                doneEdittingFile = true;    //used to escape remove or modify loop
                            }
                            else if (remOrMod.equalsIgnoreCase("MODIFY") || remOrMod.equalsIgnoreCase("M") ||
remOrMod.equalsIgnoreCase("CHANGE") || remOrMod.equalsIgnoreCase("C"))
                            {                               //interprets the modify action for the record
                                boolean finChange = false; //used for modification loop
                                while (finChange != true)
                                {
                                    System.out.println("What data within this record would you like to change?
\nforename/surname/email address/home address/post code/phone number or EXIT to leave");
                                    String changeDecision = kbReader.nextLine();//to specify what is to be changed
                                    if (changeDecision.equalsIgnoreCase("FORENAME") ||
changeDecision.equalsIgnoreCase("FIRSTNAME") || changeDecision.equalsIgnoreCase("FIRST NAME")||
changeDecision.equalsIgnoreCase("FIRST") || changeDecision.equalsIgnoreCase("F"))
                                    {                       //to change the forename
                                        System.out.println("Input new forename");
                                        String newForename = kbReader.nextLine();
                                        current = a.changeNode(current, changeDecision, newForename);
                                        System.out.println("Forename changed\n");
                                        finChange = true;  //used to escape modification loop
                                    }
                                    else if (changeDecision.equalsIgnoreCase("SURNAME") ||
changeDecision.equalsIgnoreCase("SECOND NAME") || changeDecision.equalsIgnoreCase("SECONDNAME") ||
changeDecision.equalsIgnoreCase("S"))
                                    {                       //to change the surname
                                        boolean hasSurname = false; //used for surname loop
                                        String newSurname = "";
                                        while (hasSurname != true) //test to insure surname input is not empty
                                        {
                                            System.out.println("Input New Surname");
                                            newSurname = new String (kbReader.nextLine());
                                            if (newSurname.equalsIgnoreCase("") || newSurname == null )
                                            {               //if the surname input is empty
                                                System.out.println("Invalid input - please insert a surname");
                                            }
                                            else
                                            {               //else escapes the surname loop
                                                hasSurname = true;
                                            }
                                        }
                                        current = a.changeNode(current, changeDecision, newSurname);
                                        System.out.println("Surname changed");
                                        finChange = true;
                                        a.sortList();      //sorts the list as the surname is the key value
                                    }
                                    else if ((changeDecision.equalsIgnoreCase("PHONENUMBER") ||
changeDecision.equalsIgnoreCase("PHONE NUMBER") || changeDecision.equalsIgnoreCase("PHONE") ||
changeDecision.equalsIgnoreCase("P")))
                                    {                       //to change the phone number of the record
                                        System.out.println("Input new phone number");
                                        String newNumber = kbReader.nextLine();
                                        current = a.changeNode(current, changeDecision, newNumber);
                                        System.out.println("Phone number changed\n");
                                        finChange = true;
                                    }
                                    else if ((changeDecision.equalsIgnoreCase("EMAILADDRESS") ||
changeDecision.equalsIgnoreCase("EMAIL ADDRESS") || changeDecision.equalsIgnoreCase("EMAIL") ||
changeDecision.equalsIgnoreCase("E")))
                                    {                       //to change the email address of the record
```

```
                                        System.out.println("Input new Email Address");
                                        String newEmail = kbReader.nextLine();
                                        current = a.changeNode(current, changeDecision, newEmail);
                                        System.out.println("Email address changed\n");
                                        finChange = true;
                                    }
                                    else if ((changeDecision.equalsIgnoreCase("HOMEADDRESS") ||
changeDecision.equalsIgnoreCase("HOME ADDRESS") || changeDecision.equalsIgnoreCase("HOME") ||
changeDecision.equalsIgnoreCase("H")))
                                    {                      //to change the home address of the record
                                        System.out.println("Input the new first line of house address");
                                        String newHAddress = kbReader.nextLine();
                                        current = a.changeNode(current, changeDecision, newHAddress);
                                        System.out.println("Home address changed\n");
                                        finChange = true;
                                    }
                                    else if (changeDecision.equalsIgnoreCase("POSTCODE") || changeDecision.equalsIgnoreCase
("POST CODE"))
                                    {                      //change the post code of the record
                                        System.out.println("Input new post code");
                                        String newPCode = kbReader.nextLine();
                                        current = a.changeNode(current, changeDecision, newPCode);
                                        System.out.println("Post code changed\n");
                                        finChange = true;
                                    }
                                    else if (changeDecision.equalsIgnoreCase("EXIT"))
                                    {                      //if the user does not want to edit the record
                                        finChange = true;  //exits the loop
                                        continue;          //returns to the main menu
                                    }
                                    else
                                    {                      //else if the user inputs invalid data at this stage
                                        finChange = false; //will continue in the modify loop until correct input is seen
                                        System.out.println("Invalid input - please try again");
                                    }
                                }
                                doneEdittingFile = true;   //used to escape the loop
                            }
                            else if(remOrMod.equalsIgnoreCase("EXIT") || remOrMod.equalsIgnoreCase("E"))
                            {   //if the user decides to quit instead of removing or modifying the record
                                doneEdittingFile = true;   //exit the loop - return to main menu
                                continue;
                            }
                            else
                            {                              //if invalid commands are input at the remove or modify stage
                                doneEdittingFile = false;  //continue the loop
                                System.out.println("Invalid input - please try again\n");
                                remOrMod = null;           //resets input for repeat of request when loop continues
                            }
                        }
                    }
                    else if (validation.equalsIgnoreCase("NO") || validation.equalsIgnoreCase("N"))
                    {                                      //if the record found is not the one required
                        current = result; //set it as the new current item so that the search continues from this point
                    }
                    else
                    {                                      //if an invalid command is detected
                        System.out.println("Invalid request, please try again"); //invalid input and continue loop
                    }
                }
            }
            else if(choice.equalsIgnoreCase("PRINT ALL") || choice.equalsIgnoreCase("PRINT") || choice.equalsIgnoreCase("P"))
            {                                              //if the user wants to print the list
                if (a == null || a.isEmpty() == true)
                {                                          //if the list is empty
                    System.out.println("The list is empty - cannot print list\n");
                }
                else
                {                                          //else print the list
                    a.printList(a.getHead());              //calls print list method of the class
                }
            }
            else if(choice.equalsIgnoreCase("GET SIZE") || choice.equalsIgnoreCase("SIZE") || choice.equalsIgnoreCase("GS"))
            {                                              //if the user wants the size of the list
                if (a == null || a.isEmpty() == true)
                {                                          //if the list is empty
```

```
                        System.out.println("The list is empty, the size is 0\n");
                        continue;
                    }
                    else
                    {                                           //else return and print the size of the list
                        System.out.println("The size of the list is " + a.length(a.getHead()) + " items\n");
                    }
                }
                else if(choice.equalsIgnoreCase("SORT"))
                {                                               //if the user wants to sort the list
                    a.sortList();                               //calls the sort method
                }
                else if(choice.equalsIgnoreCase("CLEAR LIST") || choice.equalsIgnoreCase("CLEAR") || choice.equalsIgnoreCase("C"))
                {                                               //if the user wants to clear the list
                    boolean doClear = false;                    //used for clear list confirmation loop
                    while (doClear != true)
                    {
                        System.out.println("Are you sure you want to clear the whole list? yes/no");
                        String toClear = kbReader.nextLine();  //confirmation
                        if (toClear.equalsIgnoreCase("YES") || toClear.equalsIgnoreCase("Y"))
                        {                                       //the user wants to clear the list
                            a = null;
                            System.out.println("List cleared\n");
                            doClear = true;                     //exit clear list loop
                            toClear = null;                     //clears input
                        }
                        else if(toClear.equalsIgnoreCase("NO") || toClear.equalsIgnoreCase("N"))
                        {                                       //the user does not want to clear the list
                            System.out.println("List not cleared");
                            doClear = true;                     //exit clear list loop
                            toClear = null;                     //clears input
                        }
                        else
                        {                                       //else the user inputs invalid command
                            System.out.println("Invalid request - please try again");
                            doClear = false;                    //exit the clear list loop
                        }
                    }
```

```
            }
            else if(choice.equalsIgnoreCase("EXIT") || choice.equalsIgnoreCase("E"))
            {                                              //if the user chooses to exit the program
                System.out.println("System closing");
                done = true;                               //exits the whole loop
            }
            else if (choice.equalsIgnoreCase("CHANGEPASSWORD") || choice.equalsIgnoreCase("CHANGE PASSWORD") ||
choice.equalsIgnoreCase("PASSWORD"))
            {
                boolean isPassword = false;                //used for change password loop
                String oldPassword = "";                   //used for password input
                while (isPassword != true)
                {
                    System.out.println("Input old password, it is case sensitive");
                    oldPassword = kbReader.nextLine();     //get old password
                    if(oldPassword.equals(getPasswordWithoutCode()))
                    {                                      //if the old password input is correct
                        System.out.println("Input new password");
                        String newPass = kbReader.nextLine();
                        setPassword(newPass);              //change password
                        System.out.println("Password Changed\n");
                        isPassword = true;
                    }
                    else
                    {                                      //if the password input is incorrect
                        boolean confirm = false;
                        while (confirm != true)
                        {   //ask user whether they want to retry input or return to main menu and not change password
                            System.out.println("Incorrect password, do you want to try again? yes/no");
                            String retry = kbReader.nextLine();
                            if (retry.equalsIgnoreCase("YES") || retry.equalsIgnoreCase("Y"))
                            {                              //if they do want to try again
                                isPassword = false;        //continue loop
                                confirm = true;            //exit this inner loop
                            }
                            else if(retry.equalsIgnoreCase("NO") || retry.equalsIgnoreCase("N"))
                            {                              //if they do not want to try again
                                isPassword = true;         //exits loop
                                confirm = true;             //exits this inner loop
                                System.out.println("Password not changed\n");
                            }
                            else
                            {                               //if the user inputs invalid command
                                System.out.println("Invalid input, please try again");
                                confirm = false;            //continues this loop until valid input is found
                            }
                        }
                    }
                }
            }
            else
            {                                               //if the user inputs wrong information at this stage
                System.out.println("Invalid request, please try again\n");
                done = false;                               //continue loop
                continue;
            }
        }
        saveListToFile(a);                                  //saves the list with any final changes
        kbReader.close();                                   //closes the reader
    }
    //-----------------------------------------------------------------------------------------------
    /**
     * This method will serialise and save the singly linked list object to the ".ser" file which is stored
     * in the 'doc' folder of this project - void, no returns
     *
     * @param lst           the Singly linked list which is to be saved
     */
    private static void saveListToFile(SinglyLinkedList lst)
    {
        try
        {
            ObjectOutputStream os = new ObjectOutputStream(new FileOutputStream(fileName));
            os.writeObject(lst);                            //writes the list to the file
            os.flush();                                     //flush the stream
            os.close();                                     //closes output stream to finalise changes to the list
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
    //----------------------------------------------------------------------------------------------
    /**
     * This method will 'deserialize' and load the SinglyLinkedList object from the '.ser' file, which
     * is saved in the "doc" folder of this project
     *
     * @return SinglyLinkedList the list which is loaded from the secondary storage
     */
    private static SinglyLinkedList loadListFromFile()
    {
        SinglyLinkedList lst = null;
        try
        {
            ObjectInputStream is = new ObjectInputStream(new FileInputStream(fileName));
            lst = (SinglyLinkedList) is.readObject();      //loads the list from the file
            is.close();                                    //closes reader
        }
        catch(FileNotFoundException e)
        {
            e.printStackTrace();
        }
        catch(IOException e)
        {
            e.printStackTrace();
        }
        catch(ClassNotFoundException e)
        {
            e.printStackTrace();
        }
        return lst;
    }
    //-----------------------------------------------------------------------------------------------
    /**
     * This method is used to set the password that the system uses as a level of security - private so
     * it cannot be changed from outside the class. It is simply a set method and it saves the password data to a new file
     *
     * @param newPassword   the new password that the user wants to change to
     */
    private static void setPassword(String newPassword)
    {
        try
        {
            ObjectOutputStream osPassword = new ObjectOutputStream (new FileOutputStream (passwordFileName));
            osPassword.writeObject(newPassword);           //writes the new password to file
            osPassword.flush();                            //flush the stream
            osPassword.close();                            //close the stream
        }
        catch (FileNotFoundException e)
        {
            e.printStackTrace();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
    //-----------------------------------------------------------------------------------------------
    /**
     * This method is used to return the user password if they forget it, it requires them to pass
     * a integer which will be compared with a unique and unchanging system code. Private so it is not
     * accessible by malicious users.
     *
     * @param systCode      to be compared with the system code
     * @return userPassword the user password returned
     */
    private static String getPassword(String systCode)
    {
        if (systCode.equals(PASSCODE))
```

```java
        {                                                       //if the passcode input is equals the system code
            String userPassword = "";
            try
            {
                ObjectInputStream isPassword = new ObjectInputStream (new FileInputStream (passwordFileName));
                userPassword = (String) isPassword.readObject();//retrieve the password from the file
                isPassword.close();                             //close the stream
                System.out.println("Correct system code, your password is: " + userPassword);
            }
            catch(FileNotFoundException e)
            {
                e.printStackTrace();
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
            catch(ClassNotFoundException e)
            {
                e.printStackTrace();
            }
            return userPassword;                                //return the retrieved password
        }
        else
        {                                                       //else if the password input is not equal to the passcode
            System.out.println("That is the incorrect system code, please try again");
            return null;
        }
    }
    //-------------------------------------------------------------------------------------------------
    /**
     * This method also returns the user password however, this method is only used to compare the
     * passwords input with those on file - it is not accessible by the user - takes no parameters
     *
     * @return userPassword the password stored on file
     */
    private static String getPasswordWithoutCode()
    {
            String userPassword = "";
            try
            {
                ObjectInputStream isPassword = new ObjectInputStream (new FileInputStream (passwordFileName));
                userPassword = (String) isPassword.readObject();//retrieve the password from the file
                isPassword.close();                             //close the stream
            }
            catch(FileNotFoundException e)
            {
                e.printStackTrace();
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
            catch(ClassNotFoundException e)
            {
                e.printStackTrace();
            }
            return userPassword;
        }
        //-----------------------------------------------------------------------------------------------
        /**
         * This is a "set" method which sets the file name of the ".ser" file which stores the data in
         * Serialized form - void as it simply sets data and private because it should not be accessible
         * from outside this class.
         *
         * @param newFileName       sets the variable fileName as this value
         */
        private static void setFileName(String newFileName)
        {
            fileName = newFileName + ".ser";                  //saves the file path of the customer database file
        }
        //-----------------------------------------------------------------------------------------------
        /**
         * This method simply returns the value assigned to fileName, returns the name of the file - simply
         * a get method so there are no parameters and private so outside users cannot see the name of the file
         *
```

```
     * @return String           the file name
     */
    private static String getFileName()
    {
        return fileName;                                       //returns the file name of the database location
    }
    //-----------------------------------------------------------------------------------------------
    //state variables - private for data protection
    private static String fileName = "ListData.ser";       //the name of the file being accessed
    private static final String PASSCODE = "14GF5602C2";   //a set passcode used to retrieve user password if lost
    private static String passwordFileName = "passDat.ser";//the string in which the user password is stored
}
```