24.11.2020

ALGORITHMS & DATA STRUCTURES

SIMPLE DATABASE PROGRAM

Aydın Güler

1. PURPOSE

The subject of the project is to write a simple database program of COM480 E-trade company. This java program will be used by COM480 computer department. The main purpose of this program is to keep a customer information and his/her trade information.

In this work, NetBeans IDE 8.2 is used as working environment.

2. THE ALGORITHM

Array

The array size cannot be dynamically changed in Java. Therefore, one of the following methods can be done to add an element to the array:

By creating a new array

By using `ArrayList` as intermediate storage

In this program, adding the customer objects to the customer array is done by using `ArrayList` as intermediate storage. In this way, there is no need to determine the size of the array in advance.

In order to use `ArrayList` as intermediate storage, a new `ArrayList` must first be created with the original array, using `asList()` method. Next, required element can be added to the list using `add()` method. Then this `ArrayList` can be converted to an array using `toArray()` method.

Singly Linked List

Linked List is a linear data structure, like arrays. However, linked list elements are not stored at contiguous location, the elements are linked using pointers. It is shown in the figure below.

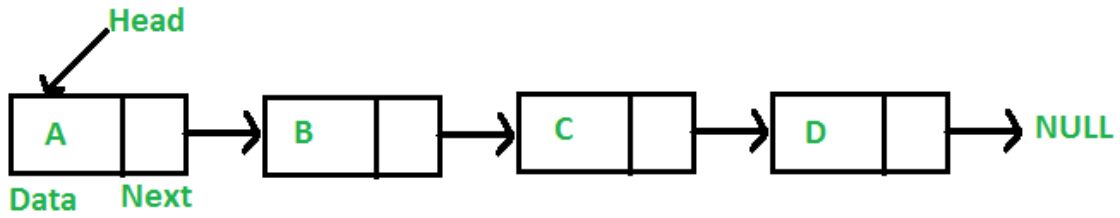

Figure 1

Figure 1 shows a linked list. A,B,C,D are the nodes and all nodes have a link.

Linked List can be represented as a class and a node can be represented as a separate class in Java. A reference to the Node class type is included in the linked list class. Node for a linked list can be represented as below:

```
Class Node {
  public int data;
  public Node link;
}
```

The following code creates a linked list class. In this class a `head` node is created. In the Constructor, `head` node assigned as null.

```
Public class singlyLinkedList {
  Private Node head;
  Public void singlyLinkedList(){
       head = null;
  }
}
```

Adding a New Element in Singly Linked List

The following `add` method creates a Node object, assigns its reference to top, initializes its data field, and assigns head to its link field:

```
Public class singlyLinkedList {
   Private Node head;
   Public void singlyLinkedList(){
        head = null;
   }
   Public void add(int data) {
        Node top = new Node();
        top.data = data;
        top.link = head;
        head = top;
   }
}
```

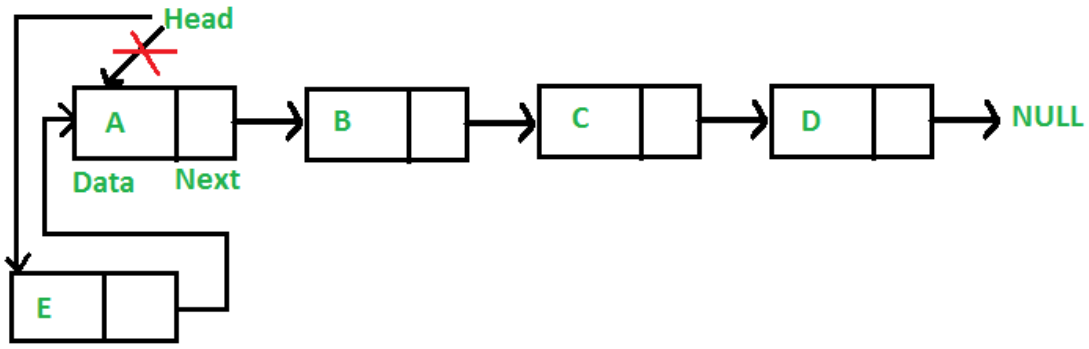

Figure 2

Figure 2 shows add method.

The new node added before the head of the given linked list. Newly added node will become the linked list's new head

3. SOLUTION

NetBeans IDE 8.2 was used to compile this project. The codes are written in Java. This report will not include the whole code that is used because of it is too long. But it will explain the variables, methods and how they are work.

The main purpose of this program is to keep a customer information and his/her trade information. In order to do that some methods are applied which are adding customer, getting the customer information, adding new item, getting total trade of customer, getting total trade of company, searching a customer, reading the trade text data from file.

The main file of the project is `SimpleDatabaseProgram.java` includes all operation codes, another file is `Customer.java` has a class to get the customer information. The other file is `Item.java` has a class that is used for determining a node with ItemName, Date, Price and a Link. `IDNotFoundException.java` is another file consists a class to give an exception for addNewItem method. The other file is named `EEM480DataBase.java` has a class called `DataBase` that has whole the processes and methods which are defined in `DataBaseInterface.java`.

DataBaseInterface.java

`DataBaseInterface.java` is an interface that specifies what will be used in this project as a method.

```
public interface DataBaseInterface {
    public void addCustomer(Customer newCustomer);
    public void listItems(int ID);
    public Customer getNewCustomer(String Name, String Surname, int ID);
    public void addNewItem (Integer ID, String ItemName, String Date, float Price);
    public Float getTotalTradeofCustomer(int ID);
    public Float getTotalTrade();
    public void readFromFile(String path);
    public Customer search_Customer(int ID);
}
```

Item.java

`Item.java` has a class called `Item` that consist `ItemName`, `Date` as a string and `Price` as a float and `Link` as an Item.

```
public class Item {
    String ItemName;
    String Date;
    float Price;
    Item Link;
}
```

Customer.java

`Customer.java` has a class called `Customer` that consist `Name`, `Surname` as a private string and `ID` as a private integer and `Link` as a private Item. Also, all of them have getters and setters in it. Thus, they can be reached outside of the class. Also, every customer's first item is created in this class and it is assigned as null.

```
public class Customer {
    private String Name;
    private String Surname;
    private int ID;
    private Item Link;

    @Override
    public String toString(){
        String retstr = "Name: " + Name;
        retstr = retstr + " Surname: " + Surname;
        retstr = retstr + " ID: " + ID;
        return retstr;
    }
    Item first;
    public Customer(){
        first = null;
```

```
    }
    public String getName() {
        return Name;
    }
    public void setName(String Name) {
        this.Name = Name;
    }
    public String getSurname() {
        return Surname;
    }
    public void setpSurname(String Surname) {
        this.Surname = Surname;
    }
    public int getID() {
        return ID;
    }
    public void setID(int ID) {
        this.ID = ID;
    }
    public Item getLink() {
        return Link;
    }
    public void setLink(Item Link) {
        this.Link = Link;
    }
  }
```

IDNotFoundException

`IDNotFoundException.java` consists `IDNotFoundException` class. This class is created to give an exception for `addNewItem` method.

```
public class IDNotFoundException extends RuntimeException{
    public IDNotFoundException(String message) {
    super(message);
    }
}
```

DataBase

`DataBase.java` is created for using `DataBase` class that consist the other methods define below.

Methods and Explanations of DataBase.java

```
public void addCustomer(Customer newCustomer)
```

Adds the customer object to the customer array. Firstly, a new array(`customerArray`) is created outside of this method. In this method a new `ArrayList` is created to keep customer object in it. Then, the `newCustomer` object is added in it. Then, this ArrayList is converted to the array(`customerArray`) which is created outside of this method. Thus, a customer array is created.

```
private Customer customerArray[] = {};
@Override
public void addCustomer(Customer newCustomer) {
        List<Customer>arrlist=newArrayList<>(Arrays.asList(custome
        rArray));
        arrlist.add(newCustomer);
        customerArray = arrlist.toArray(customerArray);
}
```

```
public void listItems(int ID)
```

Lists all the items in the list of a customer. Item `current` is created. It simply point to the first item to be printed. There is a if condition that checks whether the current item is null. Next, there is a while loop which gets into a loop until the last item to be printed. Every item will be printed in this loop. Pointer is incremented in this loop to get each item.

```
while(current != null) {

    System.out.print(current.ItemName+""+current.Date+""+current.Price
    +"\n");

    current = current.Link;

 }
```

```
Public Customer getNewCustomer(String Name, String Surname, int ID)
```

Reverse link algorithm Gets the information of new customer from user. This information contains `Name`, `Surname`, and `ID`. They are setted up with the help of setter defined in `Customer` class. This setters are `setName`, `setSurname`, `setID`.

```
Customer newCustomer = new Customer();
newCustomer.setName(Name);
newCustomer.setpSurname(Surname);
newCustomer.setID(ID);
```

```
public void addNewItem(Integer ID, String ItemName, String Date, float Price)
```

Adds the new item to the linked list of corresponding array location which contains `ID`. In this method, if `ID` is not found, then `IDNotFoundException` is thrown.

```
if(search_Customer(ID) == null) {
    throw new IDNotFoundException("Enter a Valid ID");
}
```

First, `newItem` is created. This item will keep the item information temporarly. Then, this item is added to the linked list with the help of setter(`setLink`) and getter(`getLink`) defined in `Customer` class.

```
newItem.ItemName = ItemName;
newItem.Price = Price;
newItem.Date = Date;
search_Customer(ID).setLink(newItem);
```

Here is the linking process:

```
search_Customer(ID).getLink().Link = search_Customer(ID).first;
search_Customer(ID).first = search_Customer(ID).getLink();
```

public Float getTotalTradeofCustomer (int ID)

Gets `ID` of the user and finds the total amount of expenses of her/him and returns the result. In this method, Item `current` will point to the first item. Then, a float variable(`expenses`) is defined. This variable will keep the expenses.

```
Item current = search_Customer(ID).getLink();
float expenses = 0; // Expenses will be kept within this variable.
```

There is a if statement that checks the `current` is null or not. In other words, it checks whether the customer does have item. If it is null, then it will print "List is empty".

```
if(current == null) {
   System.out.println("List is empty");
}
```

While loop checks if customer have at least 1 Item, then the item prices will be added up inside the while loop. İn order to get each item pointer is incremented. At the end it returns `expenses`.

```
while(current != null) {
      expenses += current.Price;
      current = current.Link;
}
return expenses;
```

public Float getTotalTrade ()

It almost does the same job as `getTotalTradeofCustomer`. However, this time it returns the total trade of COM480. Finds and shows the total amount of trade of the company. In this method, every customer's expenses are added to each other. All customer objects stored in the customer array are obtained with the help of for loop.

```
float totalTrade = 0;
for(int i = 0; i < customerArray.length; i++) {
     Customer allCustomers = customerArray[i];
     Item current = search_Customer(allCustomers.getID()).getLink();
           .
           .
           .
}
return totalTrade;
```

```
public void  readFromFile(String path)
```

Reads the trade text data from file. This file may contain information for a new customer or a new item. `BufferedReader` is used as reader in this method. Each content inside of the file is taken line by line with the help of `readLine`. While a line may contain a customer information, the other line may contain item information.

```
BufferedReader reader;
    try {
        reader = new BufferedReader(new FileReader(path));
        String line = reader.readLine();
```

While loop checks if the line is empty or not. At the end of the loop next line is read by `readLine` again. If the next line is empty, then `readLine` wil return null.

```
while (line != null){
        .
        .
        .
    line = reader.readLine();
}
```

There are two conditions in the while loop. One of them checks whether `charAt(0)` is letter and the other one checks whether `charAt(0)` is digit.

If it is letter, then customer information will be taken and then this customer object will be added to the customer array. Since the `getNewCustomer` method requires 3 part(`String`, `String`, `int`), the line is splitted into 3 part. First element of the `line.split[]` will be the `Name` part, second element will be the `Surname` part, third element will be the `ID` part. `Integer.parseInt` parses the string argument as a integer.

```
if (Character.isLetter(line.charAt(0))){
        Customer customerFromFile;
        customerFromFile = getNewCustomer(line.split(" 
")[0],line.split(" ")[1],Integer.parseInt(line.split(" ")[2]));
        addCustomer(customerFromFile);
}
```

If it is digit, then item information will be taken and then this item will be added to the linked list of corresponding array location which contains ID. Since the `addNewItem` method requires 4 part(`Integer`, `String`, `String`, `float`), the line is splitted into 4 part. First element of the `line.split[]` will be the `ID` part, second element will be the `ItemName` part, third element will be the `Date` part, fourth element will be the `Price` part. `Integer.parseInt` parses the string argument as an integer and `Float.parseFloat` parses the string argument as a float.

```
if (Character.isDigit(line.charAt(0))){
    addNewItem(Integer.parseInt(line.split("          ")[0]),line.split("
    ")[1],line.split(" ")[2],Float.parseFloat(line.split(" ")[3]));
 }
```

public Customer search_Customer(int ID)

Searches the corresponding array location which contains the ID. Then returns the Customer object. In order to perform that task, it uses an enhanced for loop to find the object in the array.

Here is the normal for loop:

```
for (int i = 0; i < customerArray.length; i++) {
        Customer foundCustomer= customerArray[i];
            .
            .
            .
    return foundCustomer;
}
```

They do the same job. Here is the enhanced for loop:

```
for (Customer foundCustomer: customerArray) {
            .
            .
            .
    return foundCustomer;
}
```

SimpleDatabaseProgram.java

This file is the main part of the project. It defines what is going to happen.

```
DataBase MyDataBase = new DataBase();

MyDataBase.readFromFile("d:\\MyData.txt");

Float exps = MyDataBase.getTotalTradeofCustomer(98492);

System.out.println(MyDataBase.search_Customer(98492) + " Total Expense :
" +exps);

System.out.println("The Total Trade : " +MyDataBase.getTotalTrade());
MyDataBase.listItems(98492);

Customer newc = new Customer();

newc = MyDataBase.getNewCustomer("Loki", "Mischief", 2253);
MyDataBase.addCustomer(newc);

MyDataBase.addNewItem(2253, "Scepter", "Monday", 145.8f);

MyDataBase.addNewItem(2253, "Skidbaldnir", "Saturday", 2340);

System.out.println("The Total Trade : " +MyDataBase.getTotalTrade());

MyDataBase.listItems(2253);
```

and the output will be like below.

```
run:
The content of file has been read
Name: Odin Surname: Allfather ID: 98492 Total Expense : 174.9
The Total Trade : 272.9
Name: Odin Surname: Allfather ID: 13456 Item List :
Raven Sunday 17.5
Sleipnir Saturday 54.4
Gungnir Wednesday 103.0
The Total Trade : 2758.7
Name: Loki Surname: Mischief ID: 2253 Item List :
Skidbaldnir Saturday 2340.0
Scepter Monday 145.8
```

The file MyData.txt is included into the SimpleDatabaseProgram folder.