BACS2063 Data Structures and Algorithms

ASSIGNMENT 202105

Student Name : Teh Jin Yang

Student ID : 21WMR05336

Programme : RST2

Tutorial Group : G2

Assignment Title : Online Karaoke System

Declaration

- I confirm that I have read and complied with all the terms and conditions of Tunku Abdul Rahman University College's plagiarism policy.
- I declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my own properly derived work.

yang

Student's signature

_____27/8/2021______
Date

Table of Contents

1. Introduction **3**

2. Abstract Data Type (ADT) Specification **4-5**

3. ADT Implementation **6**
3.1 Overview of ADT 6
3.2 ADT Implementation 7-12

4. Entity Classes **13**
4.1 Entity Class Diagram 13
4.2 Entity Class Implementation 14-17

5. Client Program **18-23**

1. Introduction

Project overview
Project name: TWL karaoke
Type of Client program : Online Karaoke System

The chosen application for the assignment is an online karaoke application. This application consists of Song Module, Member Module and Session Module.

For my part, I chose the member module which manages the member who used our application. To do this module, I choose sorted linked list as my adt to help me to do the function such as add(), delete(), contains(), modify(), getTotal(), isEmpty() and getIterator().

Besides, for the client java class, there are the add member, delete member , search member, modify member and generate report. Users can make their decision by key in the number to select the function they want to run based on the menu.

Lastly, there is also a member entity class to declare all the variables, get and set for all the variables. Furthermore, it has the toString method to display, equals function to check input and objects in the list and a compareTo function to compare member names for sorted order in ascending order.

2. Abstract Data Type (ADT) Specification

ADT Sorted List

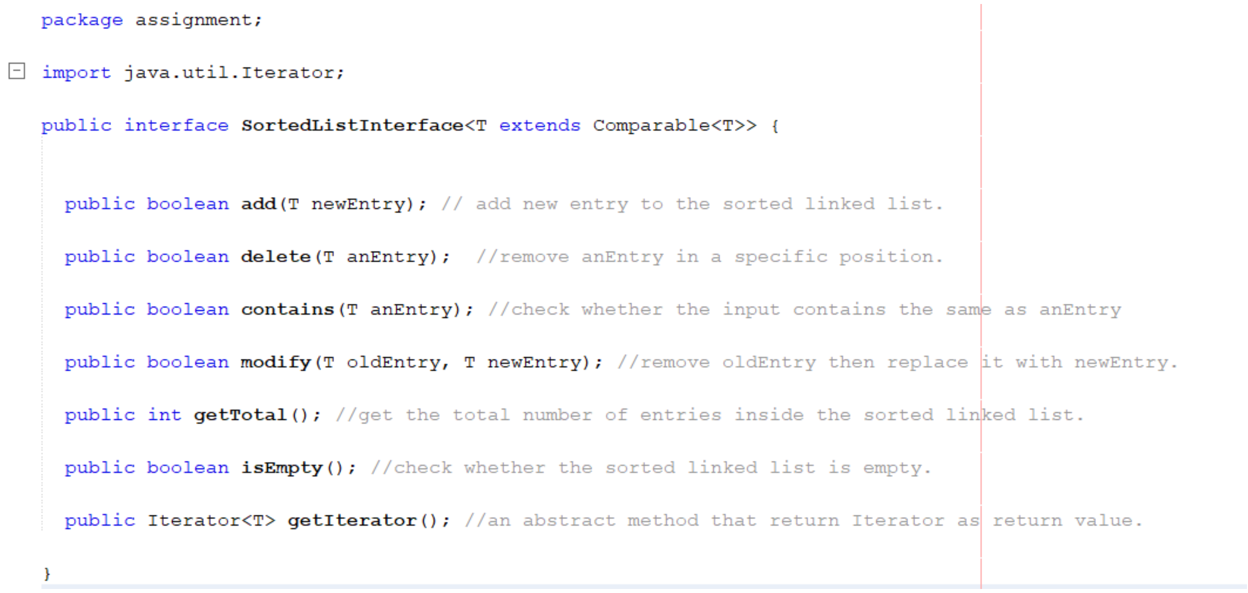
ADT Sorted List is a linear collection of entries of type T. An entry added into the list in a position that ensures the list remains sorted.

boolean add(T newEntry)	
Description:	Adds newEntry to the list that remains sorted.
Pre-condition:	-
Post-condition:	The newEntry has been added into the sorted list in a position which ensures that the list remains sorted.
Return(if any):	return true when newEntry is added else return false.
boolean delete(T anEntry)	
Description:	Removes the first or only occurrence of anEntry from the sorted list.
Pre-condition:	The list is not empty.
Post-condition:	anEntry has been removed from the list. If anEntry is not found in the list, the list remains unchanged.
Return(if any):	returned true if anEntry is located and removed, else return false.
boolean contains(T anEntry)	
Description:	Determines whether the list contains anEntry or not.
Pre-condition:	-
Post-condition:	The list remains unchanged.

Return(if any):	Return true if anEntry is located in the list, else return false.
boolean modify(T oldEntry, T newEntry)	
Description:	remove the oldEntry from the sorted list and add newEntry into the sorted list.
Pre-condition:	-
Post-condition:	oldEntry has been removed from the list and newEntry has been added into the sorted list in a position which ensures that the list remains sorted.
Return(if any):	Return true if oldEntry is located and removed and newEntry is added, else return false.
getTotal()	
Description:	Count the total number of entries currently in the list.
Pre-condition:	-
Post-condition:	The list remains unchanged.
Return(if any):	The total number of entries currently in the list.
boolean isEmpty()	
Description:	Check whether the list is empty.
Pre-condition:	-
Post-condition:	The list remains unchanged.
Return(if any):	Return true if the list is empty, else return false.
Iterator<T> getIterator()	
Description:	Get an iterator
Pre-condition:	-
Post-condition:	The list remains unchanged
Return(if any):	A new iterator.

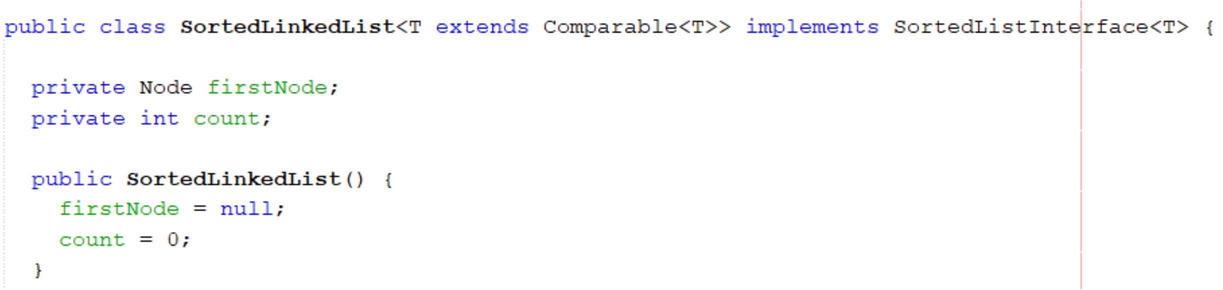
3. ADT Implementation

3.1 Overview of ADT



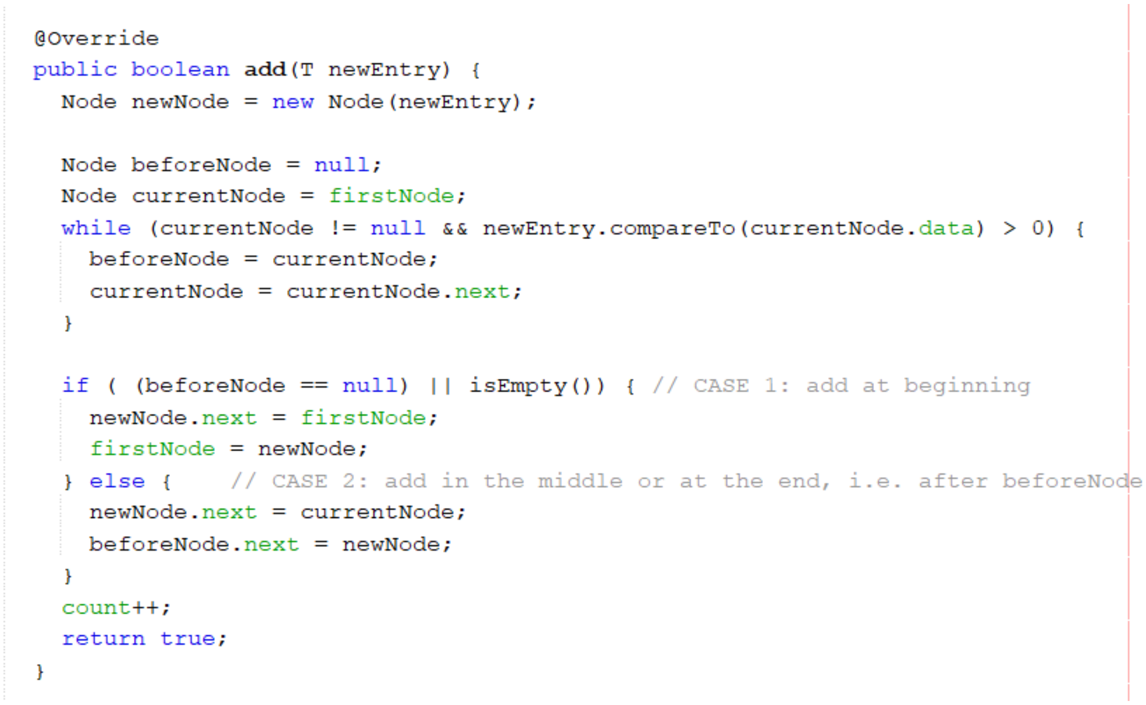
```
package assignment;

import java.util.Iterator;

public interface SortedListInterface<T extends Comparable<T>> {


  public boolean add(T newEntry); // add new entry to the sorted linked list.

  public boolean delete(T anEntry);  //remove anEntry in a specific position.

  public boolean contains(T anEntry); //check whether the input contains the same as anEntry

  public boolean modify(T oldEntry, T newEntry); //remove oldEntry then replace it with newEntry.

  public int getTotal(); //get the total number of entries inside the sorted linked list.

  public boolean isEmpty(); //check whether the sorted linked list is empty.

  public Iterator<T> getIterator(); //an abstract method that return Iterator as return value.

}
```

3.2 ADT Implementation



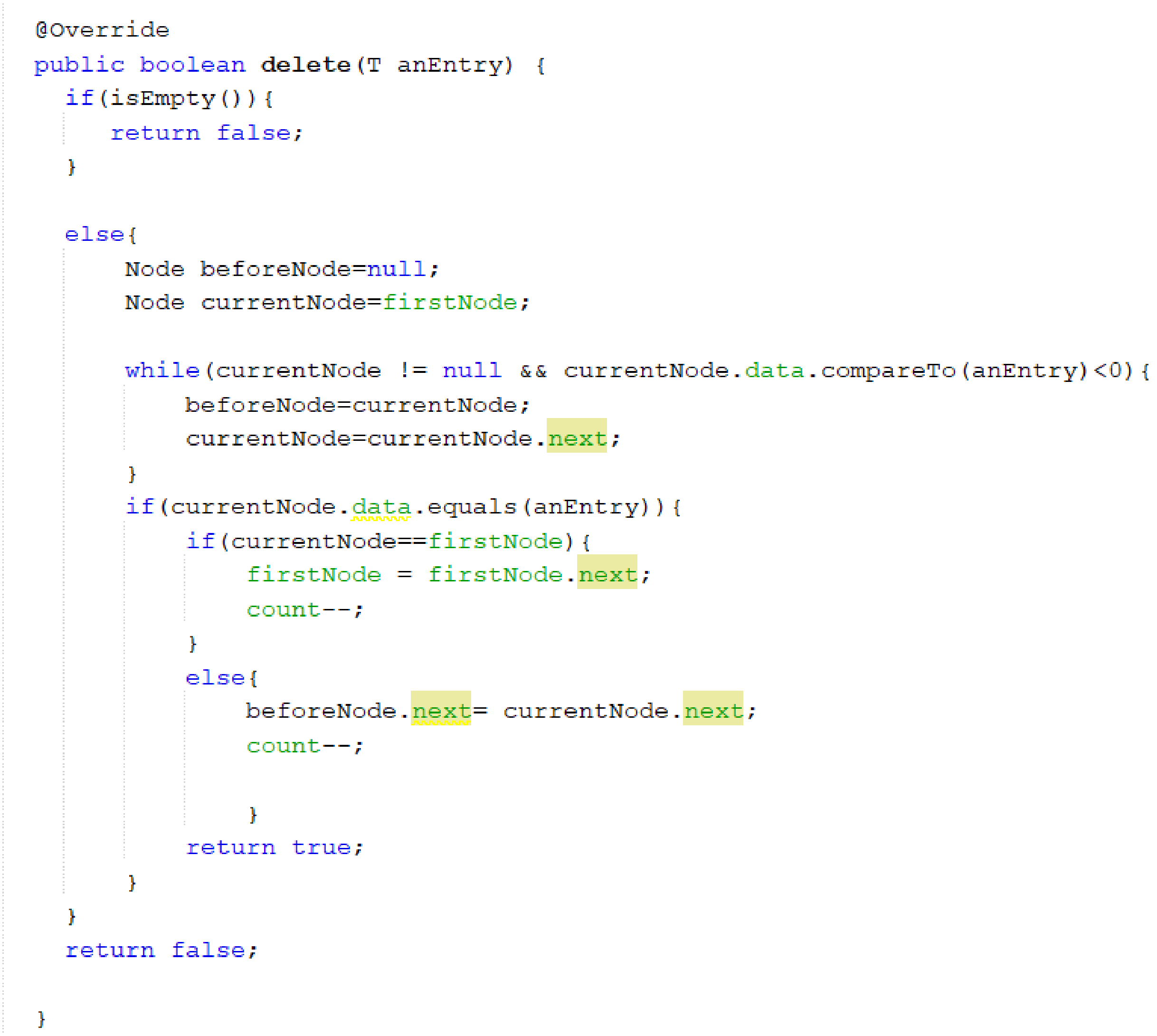
```
public class SortedLinkedList<T extends Comparable<T>> implements SortedListInterface<T> {

  private Node firstNode;
  private int count;

  public SortedLinkedList() {
    firstNode = null;
    count = 0;
  }
```

I first created a class SortedLinkedList with the comparable which allows me to implement a compareTo function to compare entries in order to determine the correct location to insert a new entry. After that, inside the function, I declare a node type first node which will be used as the pointer and an integer type count which is a variable that counts for the total number entries in the list and set both of them to null by creating the constructor.



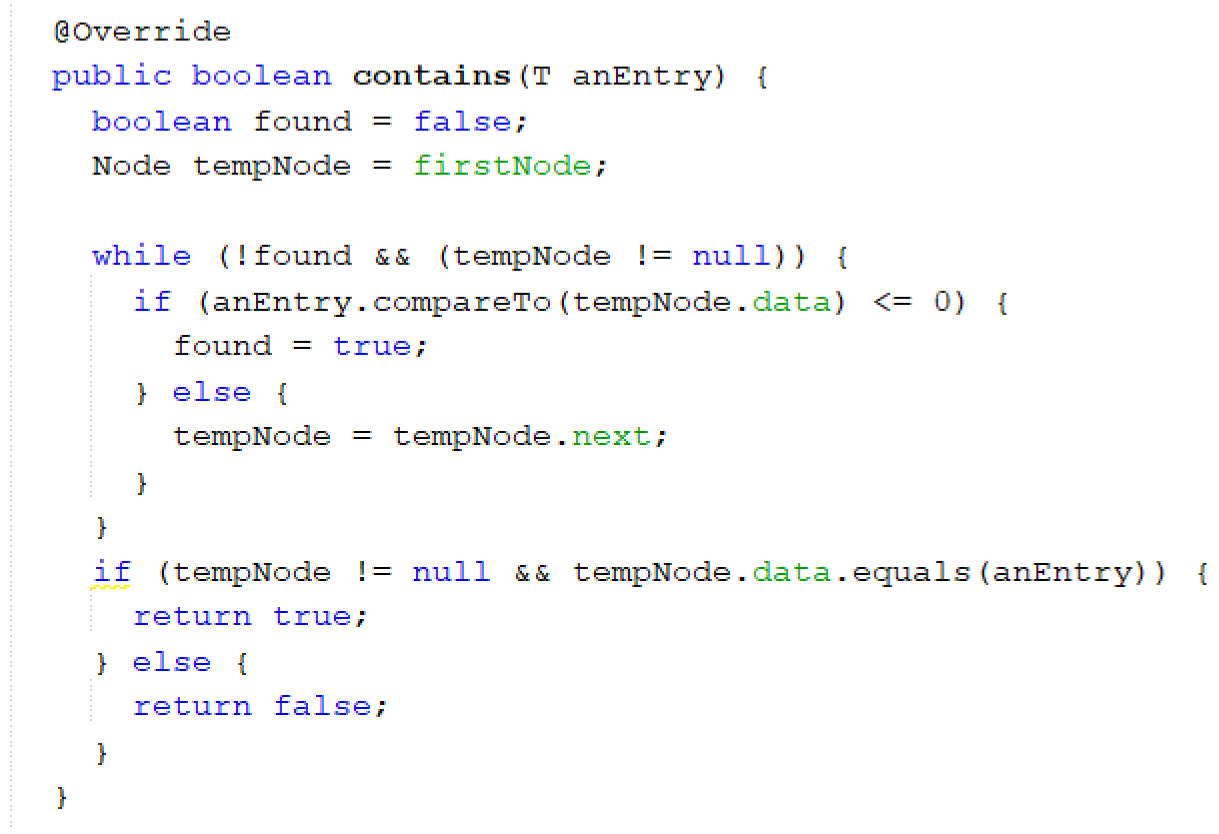
```
  @Override
  public boolean add(T newEntry) {
    Node newNode = new Node(newEntry);

    Node beforeNode = null;
    Node currentNode = firstNode;
    while (currentNode != null && newEntry.compareTo(currentNode.data) > 0) {
      beforeNode = currentNode;
      currentNode = currentNode.next;
    }

    if ( (beforeNode == null) || isEmpty()) { // CASE 1: add at beginning
      newNode.next = firstNode;
      firstNode = newNode;
    } else {    // CASE 2: add in the middle or at the end, i.e. after beforeNode
      newNode.next = currentNode;
      beforeNode.next = newNode;
    }
    count++;
    return true;
  }
```

The add(T newEntry) is to add a new entry to the sorted list. First, it will declare a beforeNode and set it to null and set the currentNode to become the firstNode. It will go into a loop when the current is not null and the newEntry compareTo currentNode data is more than 0. Inside the loop, it will make the beforeNode become the currentNode and the currentNode will become currentNode.next which is the next node and loop again. If the newEntry compareTo currentNode data is equal 0, it will not go inside the loop. It will go to the if statement which if the beforeNode equals to null, newNode.next will become the currentNode and beforeNode.next will become newNode which the newEntry will added in the beginning of the list. Else, if the beforeNode is not equal to null, newNode.next will become currentNode and beforeNode.next will become the newNode which the newEntry is added in the middle or at the end of the list. After that, count will plus one when the newEntry has been added to the list.



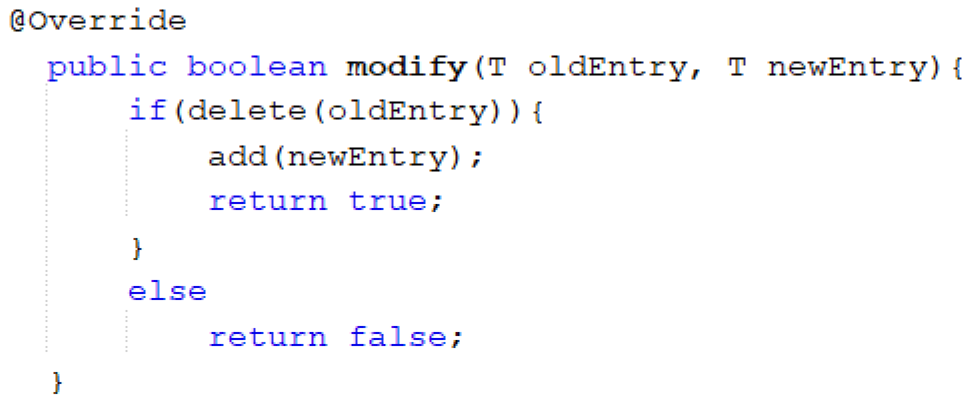
```
@Override
public boolean delete(T anEntry) {
  if(isEmpty()){
     return false;
  }

  else{
      Node beforeNode=null;
      Node currentNode=firstNode;

      while(currentNode != null && currentNode.data.compareTo(anEntry)<0){
          beforeNode=currentNode;
          currentNode=currentNode.next;
      }
      if(currentNode.data.equals(anEntry)){
          if(currentNode==firstNode){
              firstNode = firstNode.next;
              count--;
          }
          else{
              beforeNode.next= currentNode.next;
              count--;

              }
          return true;
      }
  }
  return false;

}
```

The delete(T anEntry)function will remove anEntry from the list. It will go into a loop to when the currentNode does not equal to null and currentNode.data compareTo anEntry is less than 0.Inside the loop it will set the beforeNode equals to the currentNode and set currentNode.next to currentNode. After the loop statement, it will go to the if-else statement,if the currentNode equals to firstNode, it will set firstNode to the firstNode.next which deletes the first object in the list. Else, it will set the beforeNode.next to currentNode.next and delete the object in the list according to the node.



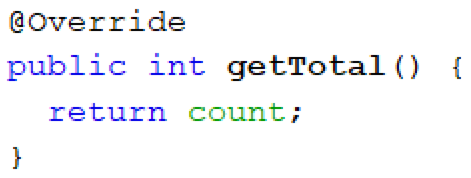
```
@Override
public boolean contains(T anEntry) {
  boolean found = false;
  Node tempNode = firstNode;

  while (!found && (tempNode != null)) {
    if (anEntry.compareTo(tempNode.data) <= 0) {
      found = true;
    } else {
      tempNode = tempNode.next;
    }
  }
  if (tempNode != null && tempNode.data.equals(anEntry)) {
    return true;
  } else {
    return false;
  }
}
```

It will determine whether the newEntry is equals to the object which is exists in the list.



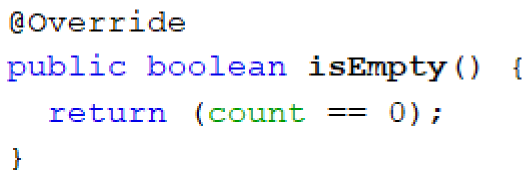
```
@Override
  public boolean modify(T oldEntry, T newEntry){
      if(delete(oldEntry)){
          add(newEntry);
          return true;
      }
      else
          return false;
  }
```

It will remove the oldEntry and add the newEntry to the list. After that, it will return true, else return false.



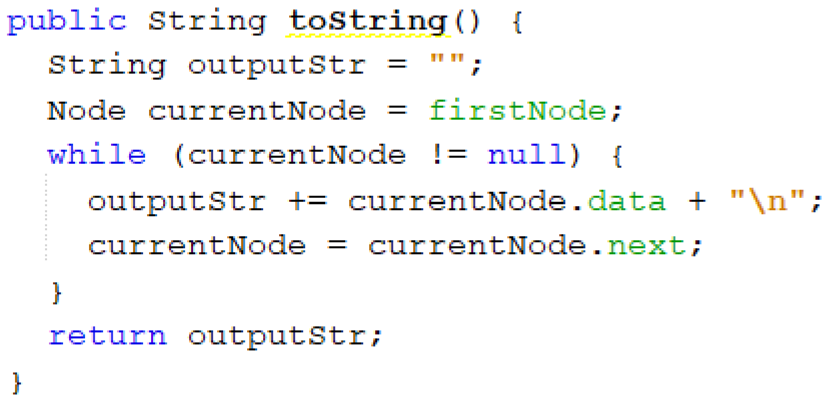
```
@Override
public int getTotal() {
  return count;
}
```

The getTotal() function is to get the total number of entries in the list. It will return the count.



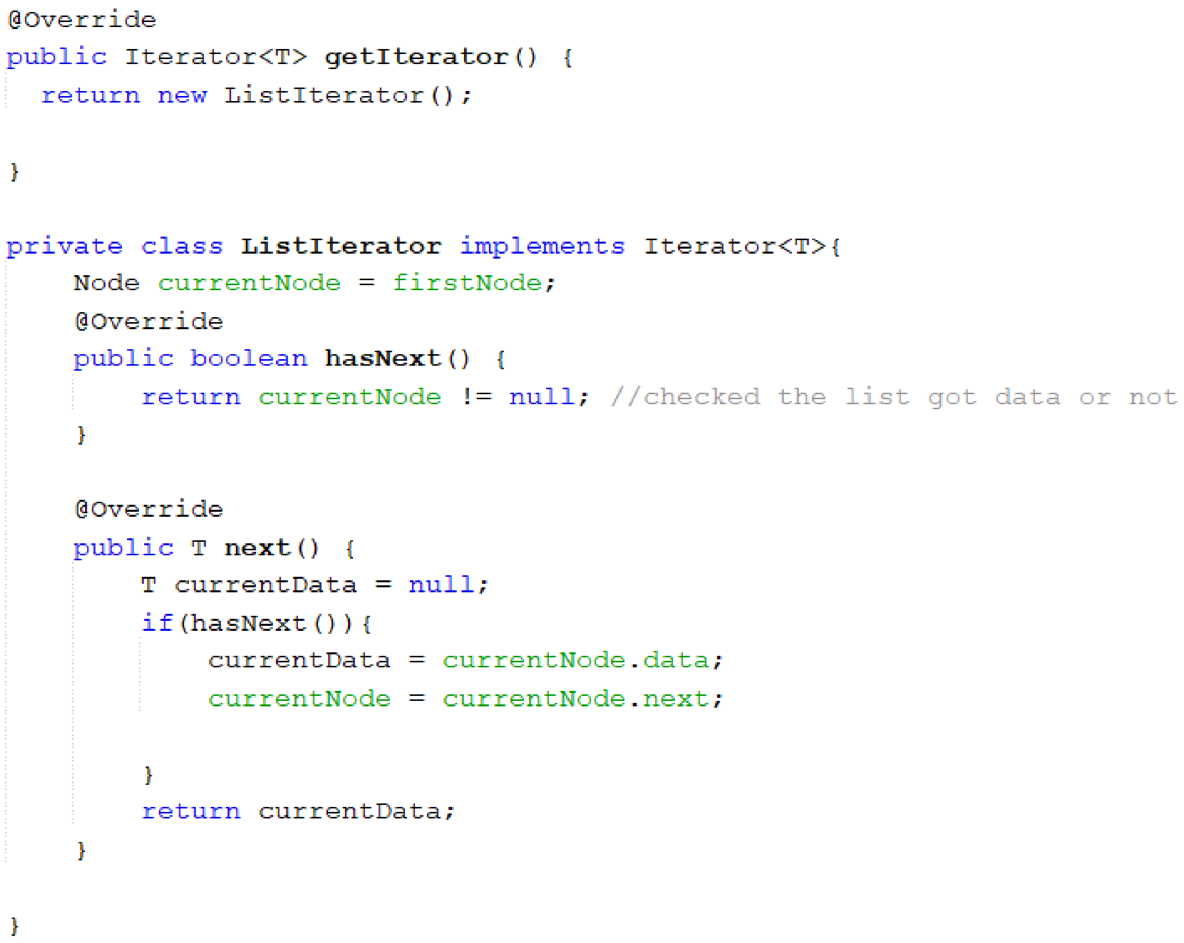
```
@Override
public boolean isEmpty() {
  return (count == 0);
}
```

It will return true if the count is 0 which means there is no have anything inside the list, else return false.



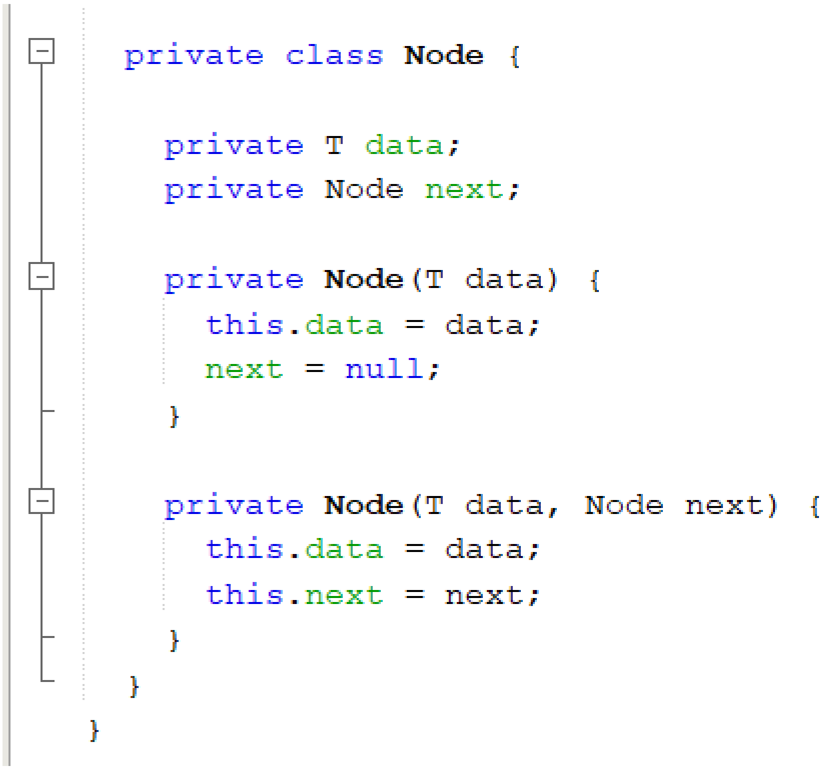
```
public String toString() {
  String outputStr = "";
  Node currentNode = firstNode;
  while (currentNode != null) {
    outputStr += currentNode.data + "\n";
    currentNode = currentNode.next;
  }
  return outputStr;
}
```

It will print out all the entries in the list based on how many total numbers of objects inside the list. It will use the currentNode as a pointer to point to the object in the list.



```
@Override
public Iterator<T> getIterator() {
  return new ListIterator();

}

private class ListIterator implements Iterator<T>{
    Node currentNode = firstNode;
    @Override
    public boolean hasNext() {
        return currentNode != null; //checked the list got data or not
    }

    @Override
    public T next() {
        T currentData = null;
        if(hasNext()){
            currentData = currentNode.data;
            currentNode = currentNode.next;

        }
        return currentData;
    }

}
```

If the currentNode does not equal to null, it will return it to the next function to let it know the list still has anything next and it needs to continue to keep looping until the currentNode in hasNext() function becomes null. The next() function will return the current node’s data and set the current node to the next node.

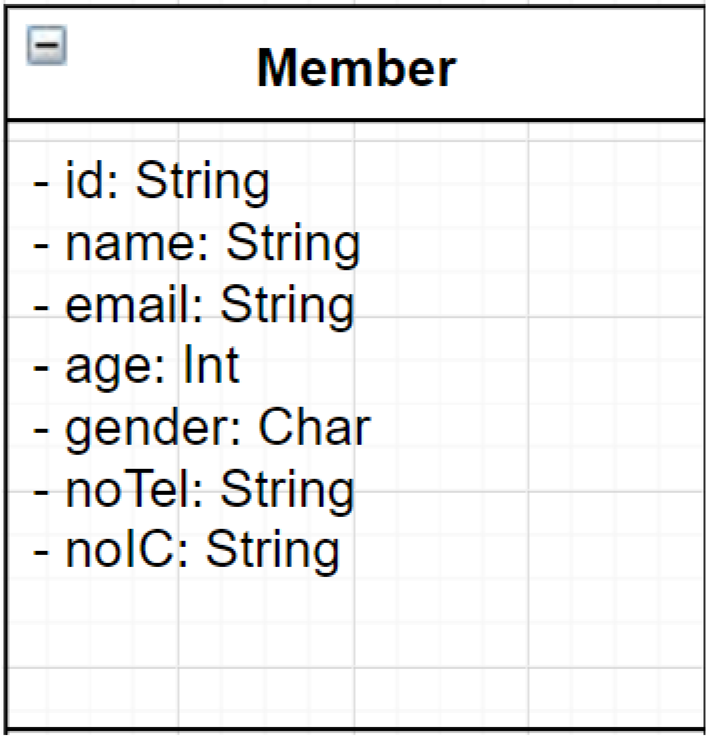


```
 private class Node {

  private T data;
  private Node next;

  private Node(T data) {
   this.data = data;
   next = null;
  }

  private Node(T data, Node next) {
   this.data = data;
   this.next = next;
  }
 }
}
```

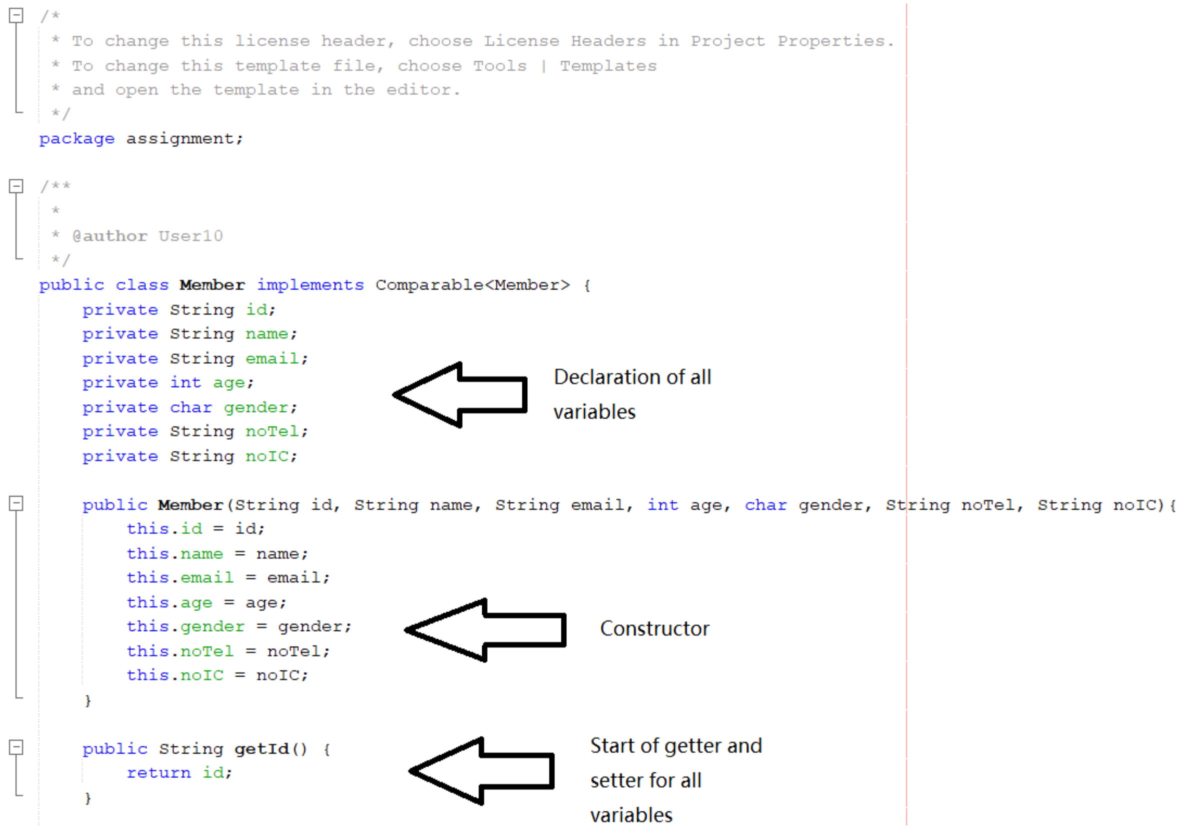
The Node class will save data and nextNode. The initial will get the data and nextNode.

4. Entity Classes

4.1 Entity Class Diagram



4.2 Entity Class Implementation

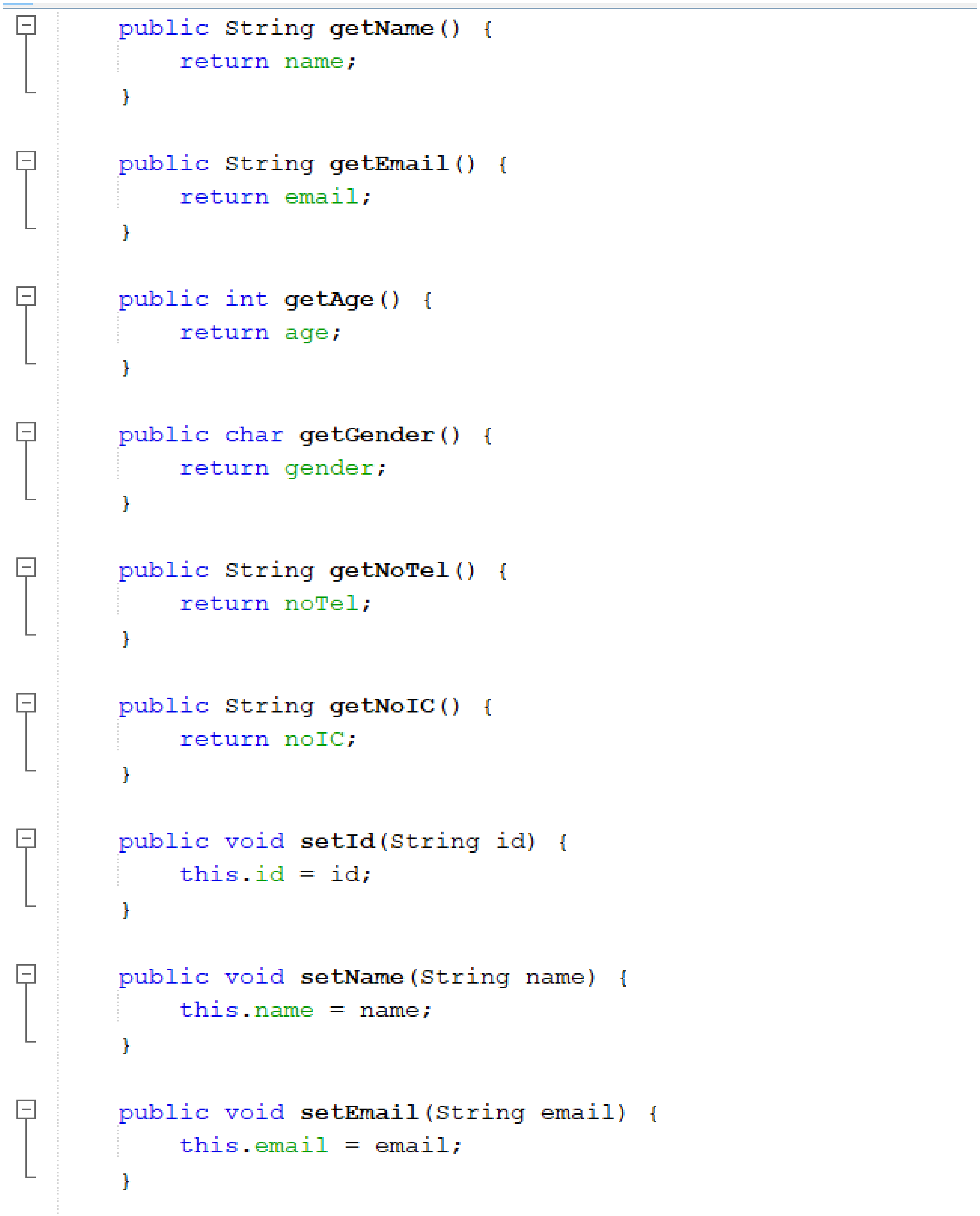


```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package assignment;

/**
 *
 * @author User10
 */
public class Member implements Comparable<Member> {
    private String id;
    private String name;
    private String email;
    private int age;
    private char gender;
    private String noTel;
    private String noIC;

    public Member(String id, String name, String email, int age, char gender, String noTel, String noIC){
        this.id = id;
        this.name = name;
        this.email = email;
        this.age = age;
        this.gender = gender;
        this.noTel = noTel;
        this.noIC = noIC;
    }

    public String getId() {
        return id;
    }
```

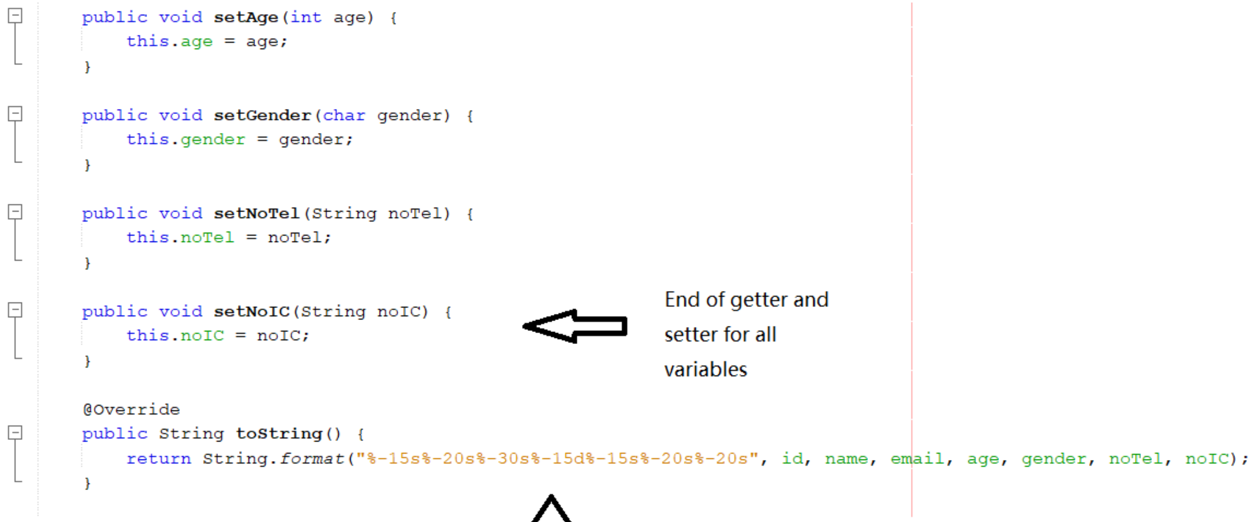
Declaration of all variables

Constructor

Start of getter and setter for all variables



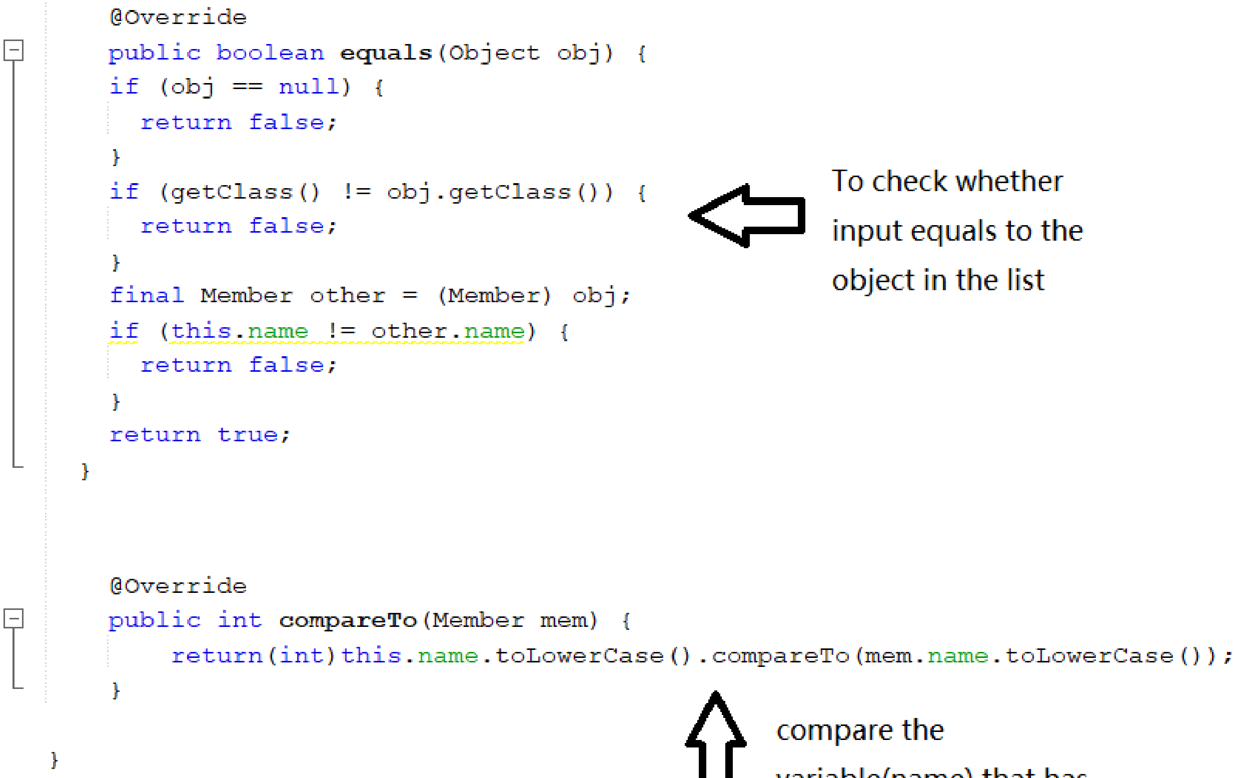
```java
    public String getName() {
        return name;
    }

    public String getEmail() {
        return email;
    }

    public int getAge() {
        return age;
    }

    public char getGender() {
        return gender;
    }

    public String getNoTel() {
        return noTel;
    }

    public String getNoIC() {
        return noIC;
    }

    public void setId(String id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setEmail(String email) {
        this.email = email;
    }
```



```java
    public void setAge(int age) {
        this.age = age;
    }

    public void setGender(char gender) {
        this.gender = gender;
    }

    public void setNoTel(String noTel) {
        this.noTel = noTel;
    }

    public void setNoIC(String noIC) {
        this.noIC = noIC;
    }

    @Override
    public String toString() {
        return String.format("%-15s%-20s%-30s%-15d%-15s%-20s%-20s", id, name, email, age, gender, noTel, noIC);
    }
```

End of getter and setter for all variables

Printing out all the variables of the class.



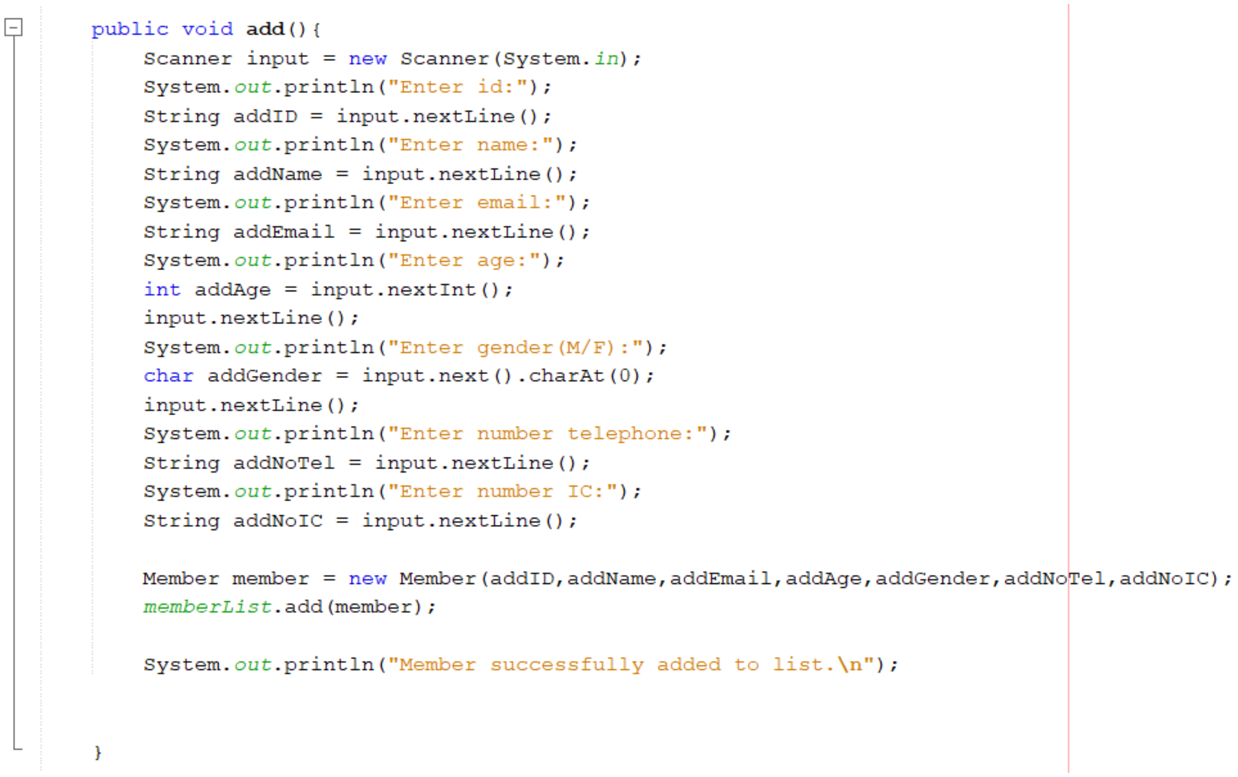
```
    @Override
    public boolean equals(Object obj) {
    if (obj == null) {
      return false;
    }
    if (getClass() != obj.getClass()) {
      return false;
    }
    final Member other = (Member) obj;
    if (this.name != other.name) {
      return false;
    }
    return true;
  }


    @Override
    public int compareTo(Member mem) {
        return(int)this.name.toLowerCase().compareTo(mem.name.toLowerCase());
    }

}
```

To check whether
input equals to the
object in the list

compare the
variable(name) that has
been assigned to the
methods to determine
the correct sort order
for the list

5. Client Program

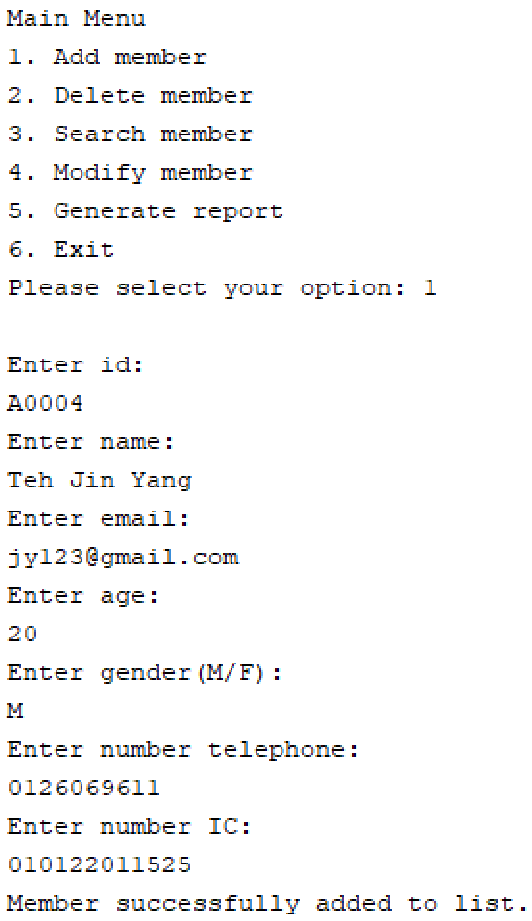
// Include a clear and concise explanation why you selected the collection ADTs that you used in your client classes.
// For console-based prototypes, include the complete source code here. For GUI-based prototypes, include clear screenshots of your code snippets illustrating the declaration and use of all collection ADTs in your client classes. Label your figures clearly.



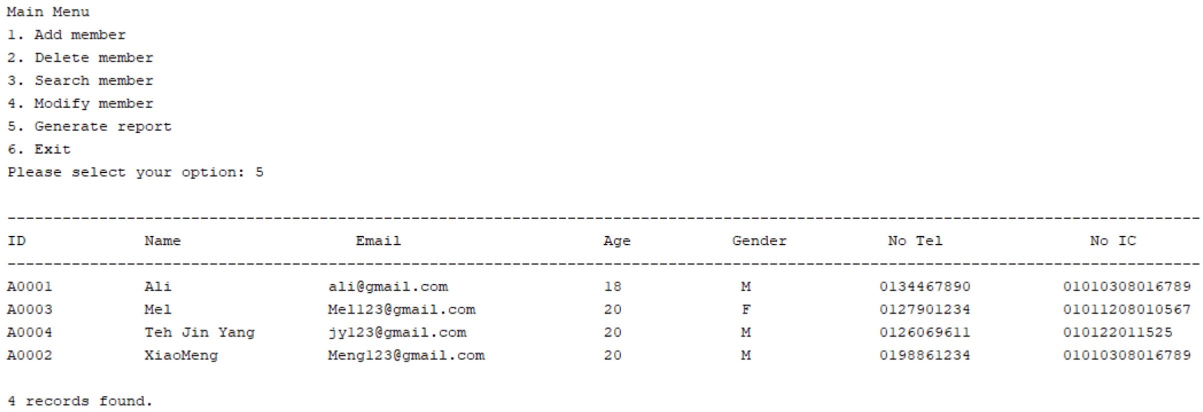
```java
    public void add(){
        Scanner input = new Scanner(System.in);
        System.out.println("Enter id:");
        String addID = input.nextLine();
        System.out.println("Enter name:");
        String addName = input.nextLine();
        System.out.println("Enter email:");
        String addEmail = input.nextLine();
        System.out.println("Enter age:");
        int addAge = input.nextInt();
        input.nextLine();
        System.out.println("Enter gender(M/F):");
        char addGender = input.next().charAt(0);
        input.nextLine();
        System.out.println("Enter number telephone:");
        String addNoTel = input.nextLine();
        System.out.println("Enter number IC:");
        String addNoIC = input.nextLine();

        Member member = new Member(addID,addName,addEmail,addAge,addGender,addNoTel,addNoIC);
        memberList.add(member);

        System.out.println("Member successfully added to list.\n");


    }
```

First, I display the message for the user to request for their input. After they finished inserting their input, I created a member type variable to get and set those inputs into each variable. After that, the system will add it to the list(memberList) by calling the add function inside the adt class.

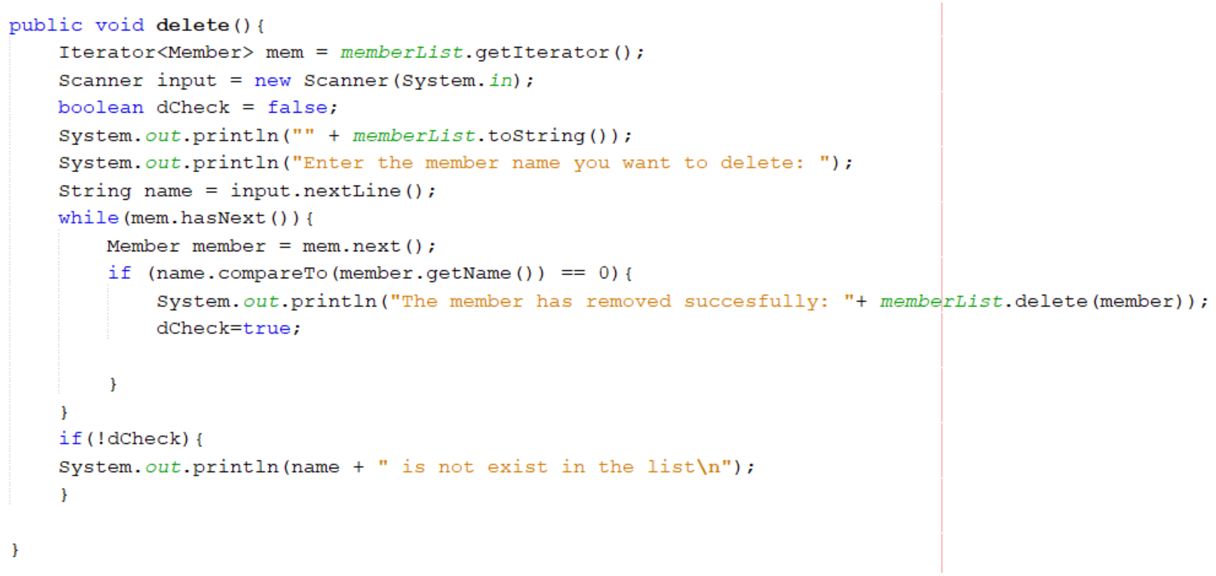
Result:



```
Main Menu
1. Add member
2. Delete member
3. Search member
4. Modify member
5. Generate report
6. Exit
Please select your option: 1

Enter id:
A0004
Enter name:
Teh Jin Yang
Enter email:
jy123@gmail.com
Enter age:
20
Enter gender(M/F):
M
Enter number telephone:
0126069611
Enter number IC:
010122011525
Member successfully added to list.
```



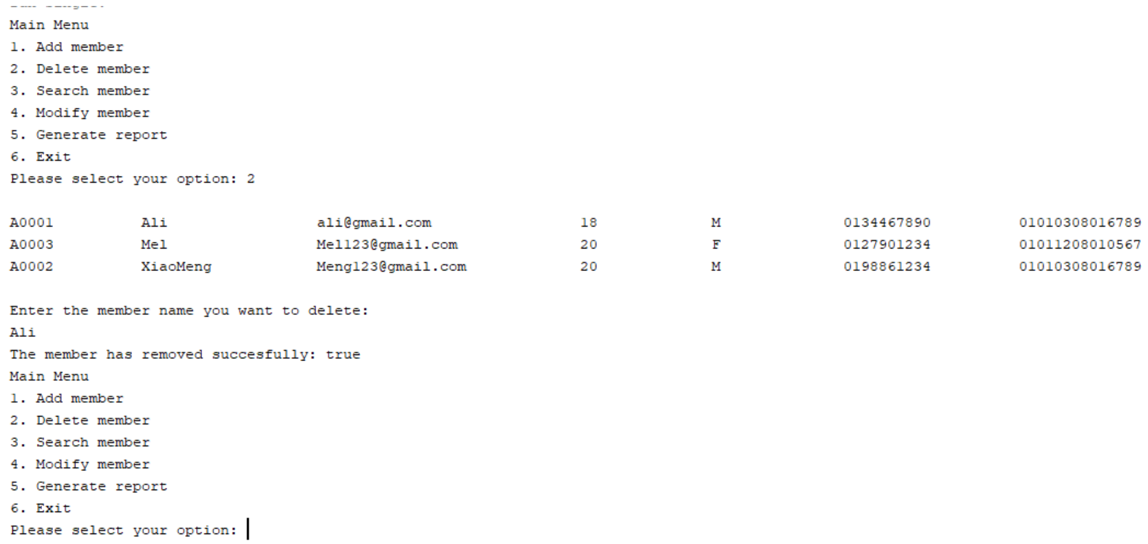
```
Main Menu
1. Add member
2. Delete member
3. Search member
4. Modify member
5. Generate report
6. Exit
Please select your option: 5

---------------------------------------------------------------------------------------------------------------------------------
ID             Name                   Email                      Age           Gender          No Tel                No IC
---------------------------------------------------------------------------------------------------------------------------------
A0001          Ali                  ali@gmail.com                18              M            0134467890           01010308016789
A0003          Mel                  Mel123@gmail.com             20              F            0127901234           01011208010567
A0004          Teh Jin Yang         jy123@gmail.com              20              M            0126069611           010122011525
A0002          XiaoMeng             Meng123@gmail.com            20              M            0198861234           01010308016789

4 records found.
```



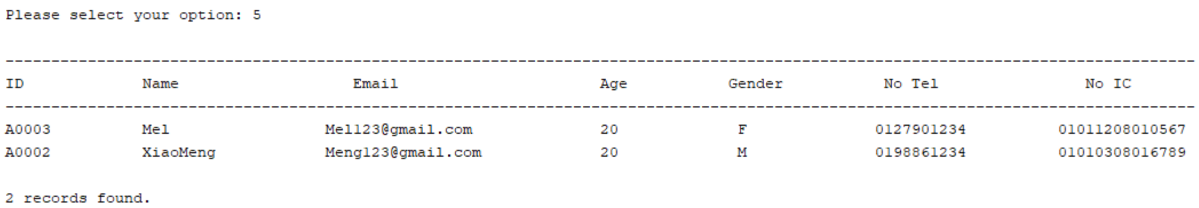
```
public void delete(){
    Iterator<Member> mem = memberList.getIterator();
    Scanner input = new Scanner(System.in);
    boolean dCheck = false;
    System.out.println("" + memberList.toString());
    System.out.println("Enter the member name you want to delete: ");
    String name = input.nextLine();
    while(mem.hasNext()){
        Member member = mem.next();
        if (name.compareTo(member.getName()) == 0){
            System.out.println("The member has removed succesfully: "+ memberList.delete(member));
            dCheck=true;

        }
    }
    if(!dCheck){
    System.out.println(name + " is not exist in the list\n");
    }

}
```

First, I declare a variable(mem) which is the Iterator type. Then, the system will display a message to request the input(name) they want to delete from the user.After that, the system will call the hasNext() and next() function which in the adt class to check if the list got object or not and get all the items' data inside the list. When the function gets one object, that object's data(name) will be compared to the input(name) inserted by the user to see whether it is the same or not. If it is the same, the system will call the delete() function in the adt class to remove the selected member out of the list. If all the objects' data(name) in the list are not the same as input(name), the system will display an error message.

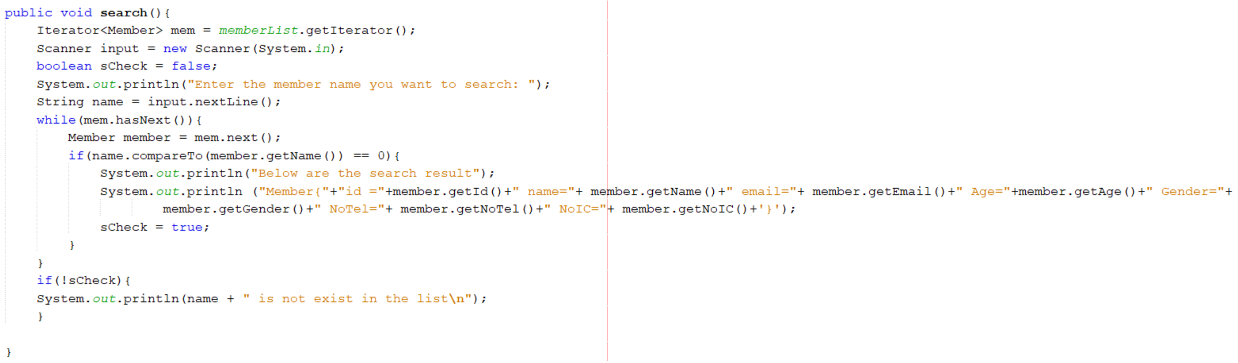
Result:



```
Main Menu
1. Add member
2. Delete member
3. Search member
4. Modify member
5. Generate report
6. Exit
Please select your option: 2

A0001          Ali                 ali@gmail.com                 18          M          0134467890          01010308016789
A0003          Mel                 Mel123@gmail.com              20          F          0127901234          01011208010567
A0002          XiaoMeng            Meng123@gmail.com             20          M          0198861234          01010308016789

Enter the member name you want to delete:
Ali
The member has removed succesfully: true
Main Menu
1. Add member
2. Delete member
3. Search member
4. Modify member
5. Generate report
6. Exit
Please select your option:
```



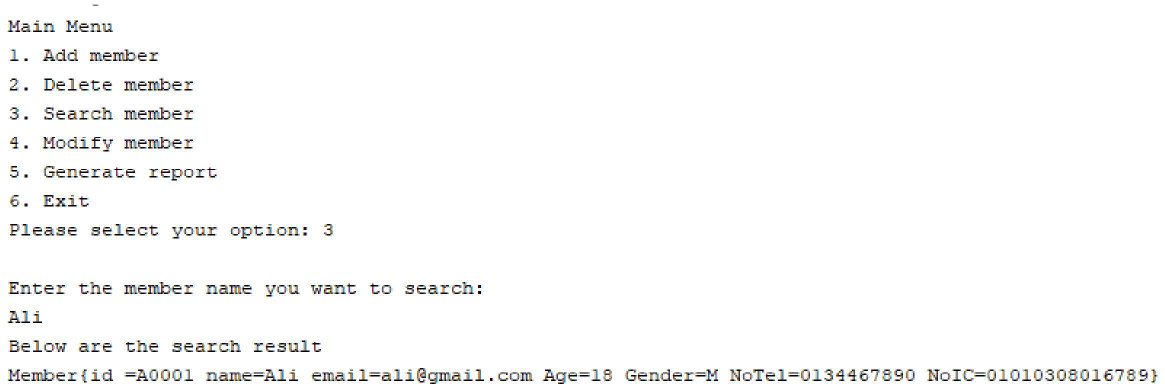
```
Please select your option: 5

-------------------------------------------------------------------------------------------------------------------------
ID             Name                 Email                    Age          Gender          No Tel              No IC
-------------------------------------------------------------------------------------------------------------------------
A0003          Mel                 Mel123@gmail.com          20             F            0127901234          01011208010567
A0002          XiaoMeng            Meng123@gmail.com         20             M            0198861234          01010308016789

2 records found.
```



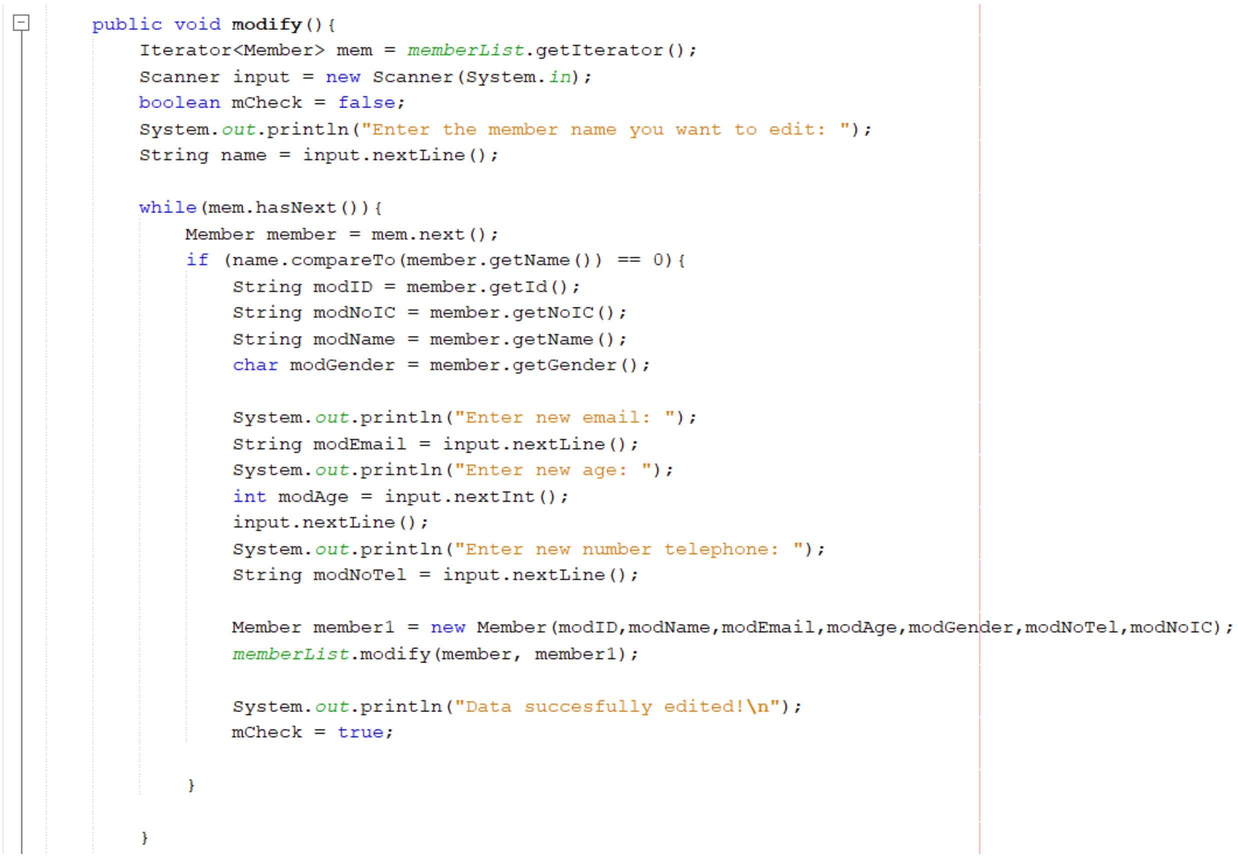
```
public void search(){
    Iterator<Member> mem = memberList.getIterator();
    Scanner input = new Scanner(System.in);
    boolean sCheck = false;
    System.out.println("Enter the member name you want to search: ");
    String name = input.nextLine();
    while(mem.hasNext()){
        Member member = mem.next();
        if(name.compareTo(member.getName()) == 0){
            System.out.println("Below are the search result");
            System.out.println ("Member{"+"id ="+member.getId()+" name="+ member.getName()+" email="+ member.getEmail()+" Age="+member.getAge()+" Gender="+
                    member.getGender()+" NoTel="+ member.getNoTel()+" NoIC="+ member.getNoIC()+'}');
            sCheck = true;
        }
    }
    if(!sCheck){
    System.out.println(name + " is not exist in the list\n");
    }

}
```

Same with delete() function above, the system will first declare a variable(mem) Iterator type. After getting the user's input(name) they want to search, the system will call hasNext() and next() from adt class to check if the list got object or not and get the data. When one object's data(name) in the list is the same as the input(name), the system will only print the selected object's data. If all the objects' data(name) in the list are not the same as input(name), the system will display an error message.

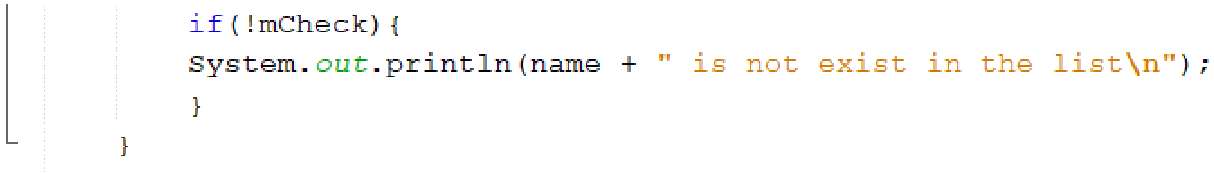
Result:



```
Main Menu
1. Add member
2. Delete member
3. Search member
4. Modify member
5. Generate report
6. Exit
Please select your option: 3

Enter the member name you want to search:
Ali
Below are the search result
Member{id =A0001 name=Ali email=ali@gmail.com Age=18 Gender=M NoTel=0134467890 NoIC=01010308016789}
```

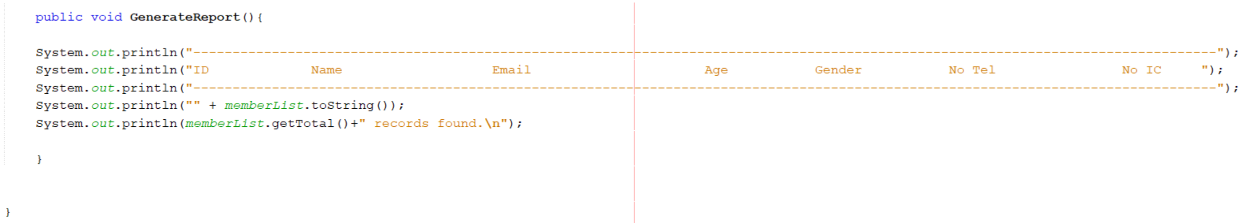


```
public void modify(){
    Iterator<Member> mem = memberList.getIterator();
    Scanner input = new Scanner(System.in);
    boolean mCheck = false;
    System.out.println("Enter the member name you want to edit: ");
    String name = input.nextLine();

    while(mem.hasNext()){
        Member member = mem.next();
        if (name.compareTo(member.getName()) == 0){
            String modID = member.getId();
            String modNoIC = member.getNoIC();
            String modName = member.getName();
            char modGender = member.getGender();

            System.out.println("Enter new email: ");
            String modEmail = input.nextLine();
            System.out.println("Enter new age: ");
            int modAge = input.nextInt();
            input.nextLine();
            System.out.println("Enter new number telephone: ");
            String modNoTel = input.nextLine();

            Member member1 = new Member(modID,modName,modEmail,modAge,modGender,modNoTel,modNoIC);
            memberList.modify(member, member1);

            System.out.println("Data succesfully edited!\n");
            mCheck = true;

        }

    }
```



```
        if(!mCheck){
        System.out.println(name + " is not exist in the list\n");
        }
    }
```

First, the system will first declare a variable(mem) Iterator type. After getting the user's input(name) they want to edit, the system will call hasNext() and next() from adt class to check if the list got object or not and get the data. When one object's data(name) in the list is the same as the input(name), the system will request the user to insert every input for each data. After getting the new data, the system will call the modify() function from the adt class to add the modified data into the sorted linked list. If all the objects' data(name) in the list are not the same as input(name), the system will display an error message.



```
    public void GenerateReport(){

    System.out.println("----------------------------------------------------------------------------------------------------------------------------------");
    System.out.println("ID             Name                   Email                      Age           Gender           No Tel                No IC     ");
    System.out.println("----------------------------------------------------------------------------------------------------------------------------------");
    System.out.println("" + memberList.toString());
    System.out.println(memberList.getTotal()+" records found.\n");

    }


}
```

The system will call the toString() function from the adt class to print all the objects' data inside the sorted linked list. Besides, the system will also call the getTotal() function from the adt class to get the total records that have been printed out.