UNIVERSITY OF
WEST LONDON

COMPONENT SOFTWARE DEVELOPMENT

Applied Software Engineering – Coursework 3

Omar Said Ibrahim – 21354926

Table of Contents

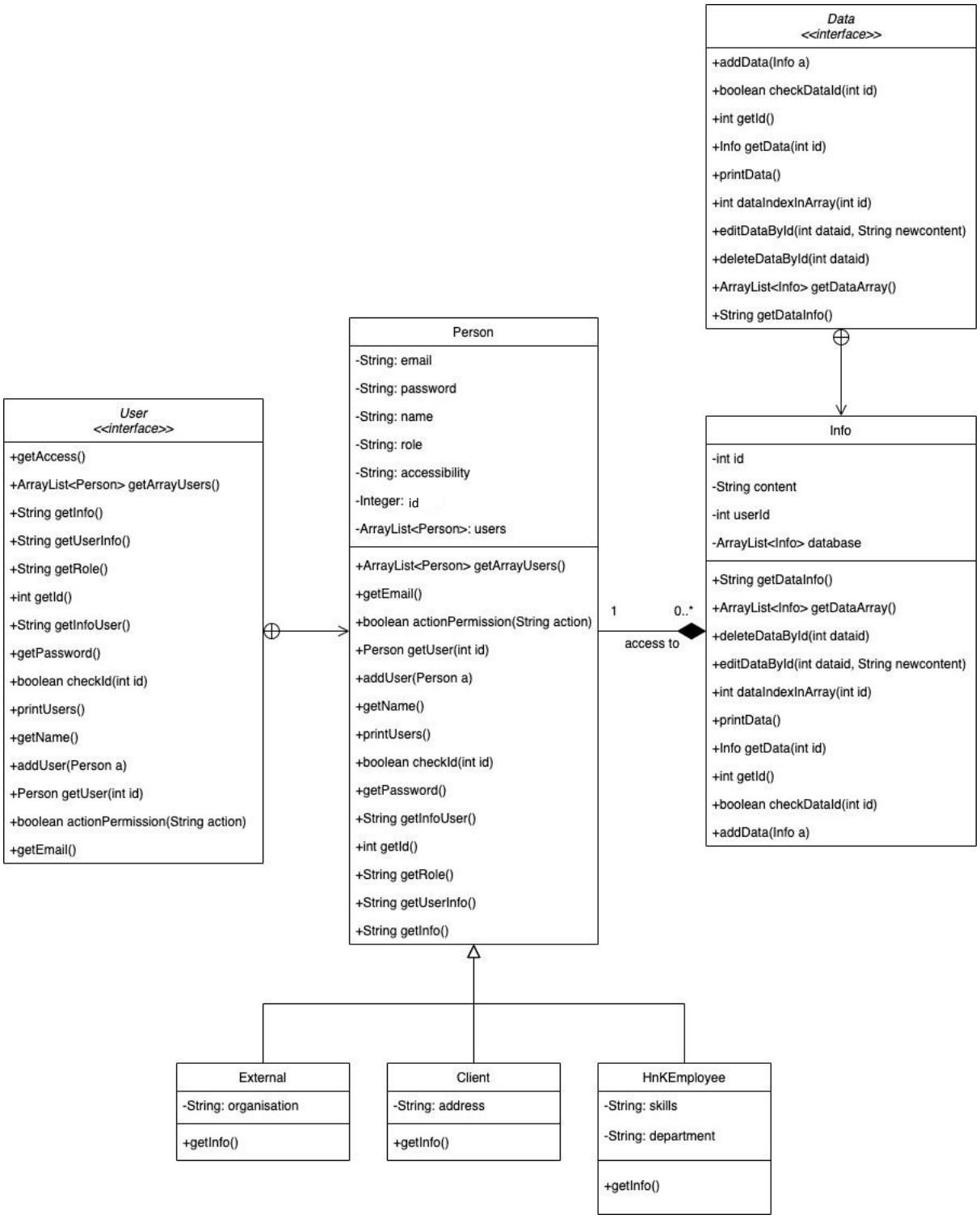
Use Case

“H&K is a global public relations firm headquartered in New York City with 1,900 employees and 68 offices in 34 countries around the world. As in so many other fields today, knowledge is central to public relations. Moreover, because public relations firms are project-centred, their employees and their clients must constantly share large amounts of information. Specifically, H&K employees need to be able to share information not only with their project mates but also with other H&K employees who have experience working with similar clients or experience working on similar products. Then they also must continually exchange information with their clients. In addition, all parties often need external information, such as market data, which they want to be able to access through the company's knowledge management system. Finally, the company needs to preserve its organisational memory, the collective, stored learning of the firm, to minimise the impact when employees leave. This memory is not only critical for H&K employees to carry out their daily work, but it is also vital in training new employees quickly and effectively.”

Implementation

The below class diagram has been slightly changed from the previous coursework. Improvements have been made: the previous class "*Dataset*" containing only the arraylist of data has been now removed and included in the class "*Info*".

Class diagram



Class diagram description

External

The class External is an extended class of “Person”. It represents a type of user, containing so all the attributes of the superclass plus another attribute called “organisation” as in the case study the organisation of the external users must be known. It also contains a method called “getInfo()” which helps to retrieve the info of the user, by using the super method of the super class (“Person”) to retrieve also the other attributes.

Client

The class Client is also an extension of the class “Person”. It contains the attributes “address” that differs from the other classes. The method “getInfo()” has the same functionalities of the other subclasses.

HnKEmployee

The class HnKEmployee represents the staff of the company as described in the use case. It in facts specifies the user skills to help Clients to find the right employee for their projects, plus the department they are from.

Person

The class Person is the super class that represents the generic user. With this class, I am able to identify the type of user logged into the program and then display the permitted actions such as print data, share data, insert data, delete data or edit data. If a user wishes to register, the class is able to add a new user to the ArrayList of Person, as well as checking the existence of a user at the time to log in. At the end of the software execution the class will return all the users and save into a file in the main class.

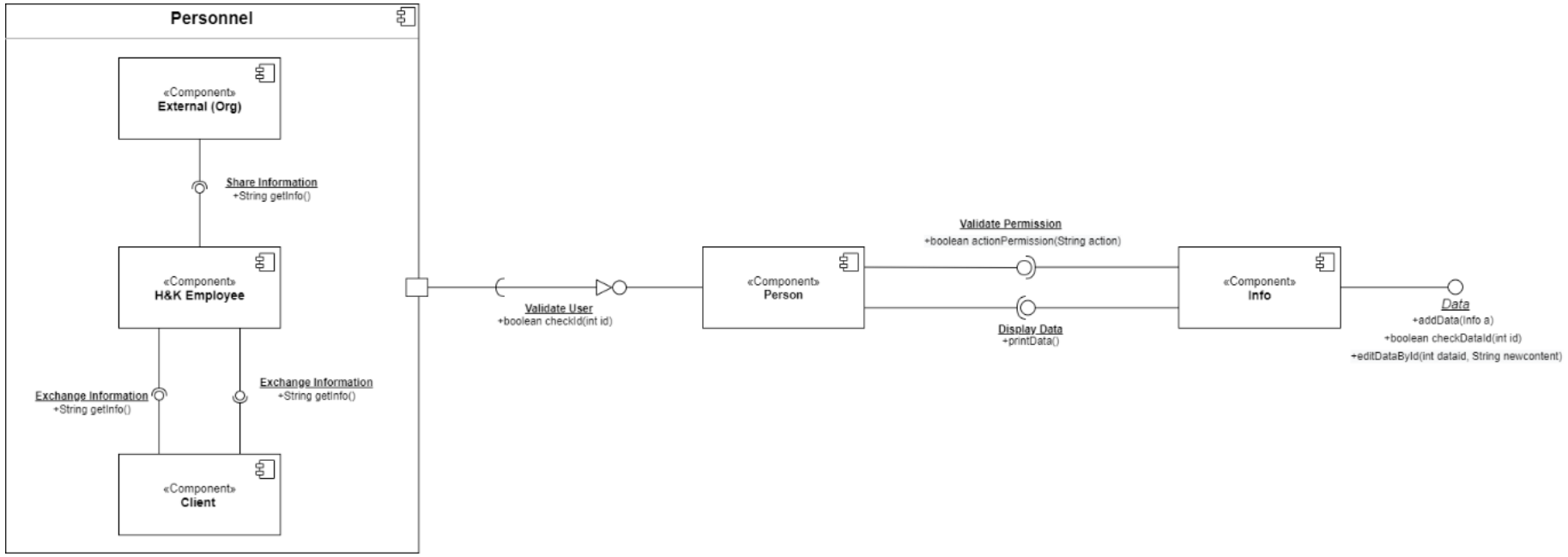
Info

The Info class contains all the data that the user inserts. All the Data collected will be saved into an external file inside the project folder. In the main class another external file called “sharedata.txt” will contains the data shared between the users.

Interfaces

The interface classes “User” and “Data” represents the methods to be implemented in the corresponding classes “Person” and “Info”.

Component model



The model above illustrates the components built from the class diagram. The Interface is represented with a lollipop and a socket shape which shows whether the component is receiving or providing the input/materials, representing the flow of data between the components. The lollipop is the circle which presents the provided interface, and the semi-circle presents the required interface.
In the picture, the personnel component contains 3 types of users: External (Org), H&K Employee and Client. The H&K employee needs to continually exchange information with the client which is done by the “getInfo” method and receive each other’s data through the same method. Similarly, H&K employee provides information to the External component through the same method also.
The personnel component has an inheritance relationship with the Person component where the personnel acquire the properties (methods and fields) of the Person component. In order to access the data, the user need to be registered; the “checkId” method ensures this. After the user been validated, they need to get permission to print the data by the “actionPermission” method. Once they have given permission, the info component allows the user to print the data by the “printData” method. Lastly, the Info component provides an interface with managing the data such as editing, adding, and checking data etc with its own methods such as adding(+addData(Info a)).

Software testing, documentation, and verification considerations

How to run the code

NOTE

The text files included in the "src.zip" ("users.txt", "database.txt" and "sharedata.txt") must be imported in the project folder. I will include the whole project folder as well just in case.

TERMINAL EXECUTION

The code can be run on terminal by running the command below in the terminal (I am using "sudo" because I am on MacOS. Remove "sudo" if you are running on Windows).

java -jar "PATH~/Coursework3_Omar/dist/Coursework3_Omar.jar"

NETBEANS

On Netbeans, we need to import the project by clicking on "file" and "import", then select the project folder.

```
Users registered:
omar@test.com-test1-omar-574-Client-Share data.Print data-St.mary
nick@test.com-test2-nick-983-External-Print data-UWL
nasser@test.com-test3-nasser-264-HnKEmployee-Share data.Print data.Edit data by ID.
Insert new data.Delete data by ID.-communication,teaching,engineering-IT

Data registered:

No data registred.


Data shared:

No data shared.


Type 'L' to Log in or 'R' to Register as a new user:
```

The first thing, the software displays all the users, all the data and all the data that have been shared everything from the external files. In this case I have inserted dummy users and I am going to show later on how data are going to be inserted as well as shared data.

Then, a user has a choice to Log in (by typing "L") or to Register (by typing "R").

Everything will be checked by the IDs that are randomly generated.

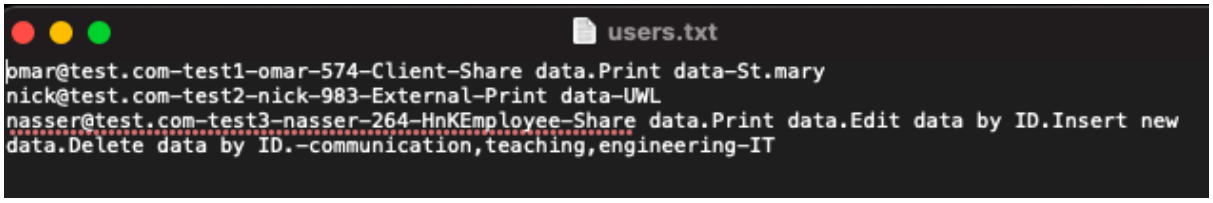
The files must not be moved from the original PATH. The files are situated inside the project folder.

```
Type 'L' to Log in or 'R' to Register as a new user: L

LOGIN

Insert ID: 574
User ID do exist

Hello!

These are your details.

ID: 574
Name: omar
Email: omar@test.com

You are a: Client

Here your possible actions:
Share data
Print data
```

At the time of the login, the software shows the details and the action permitted of the user. It will then ask which action the user would perform (in this case the actions are “Share data” or “Print data”).

```
Here your possible actions:
Share data
Print data

Choose an option or type 'E' to exit:
Print

No data to be displayed.

Choose an option or type 'E' to exit:
E

Saving everything...

Press any button to continue or type 'T' to terminate the program.
T
BUILD SUCCESSFUL (total time: 13 minutes 3 seconds)
```

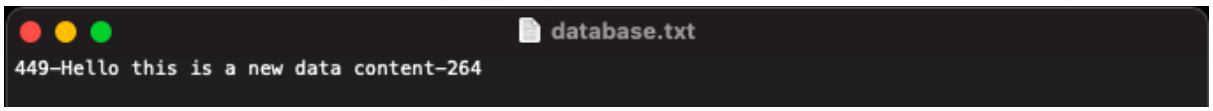
There are no data in the external file, so the software clearly does not display anything. Below, the user file where all the user info are stored.



I am going now to log in with the HnKEmployee user (id is “264”), which is the only type of user that has access to managing data.

```
LOGIN

Insert ID: 264
User ID do exist

Hello!

These are your details.

ID: 264
Name: nasser
Email: nasser@test.com

You are a: HnKEmployee

Here your possible actions:
Share data
Print data
Edit data by ID
Insert new data
Delete data by ID.

Choose an option or type 'E' to exit:
Insert
Enter content of the data:
Hello this is a new data content
New data inserted.

Choose an option or type 'E' to exit:
E

Saving everything...

Press any button to continue or type 'T' to terminate the program.
T
BUILD SUCCESSFUL (total time: 2 minutes 25 seconds)
```

The data are now saved into the file.



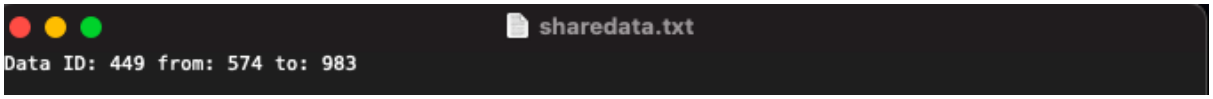
If we run the software again, the new data will be displayed at the beginning and users are now able to see them when performing the action “Print data”.

```
Users registered:
omar@test.com-test1-omar-574-Client-Share data.Print data-St.mary
nick@test.com-test2-nick-983-External-Print data-UWL
nasser@test.com-test3-nasser-264-HnKEmployee-Share data.Print data.
hing,engineering-IT

Data registered:
449-Hello this is a new data content-264

Data shared:

No data shared.


Type 'L' to Log in or 'R' to Register as a new user: L

LOGIN

Insert ID: 574
User ID do exist

Hello!

These are your details.

ID: 574
Name: omar
Email: omar@test.com

You are a: Client

Here your possible actions:
Share data
Print data

Choose an option or type 'E' to exit:
Print

Data ID: 449
Data content: Hello this is a new data content
User ID: 264
```

The user is able to share that data with another user.

```
Choose an option or type 'E' to exit:
Share
Enter ID of the data to be shared:
449
Enter ID of the User receiver:
983
Data shared successfully

Choose an option or type 'E' to exit:
E

Saving everything...

Press any button to continue or type 'T' to terminate the program.
T
BUILD SUCCESSFUL (total time: 2 minutes 16 seconds)
```

And it will be available on file and on execution.



These are the main actions the users are able to do. Everything is based on id searching; with this identification the user with the right permissions is able to delete, edit and insert a new data. Very similar is the login which does an id-search on the ArrayList of users.

OCL

Person.java

context Person::checkID(id)
pre: self.users -> isEmpty = true (p: Person l p.users = users) - - return true if the collection is empty
post: self.users -> exists (p: Person l p.users = users) - - return false if the user ID exists in the Person class

```java
//the below method returns false if the id in the parameter exists in the array, true if it does not exist
@Override
//OCL
public boolean checkId(int id){ //false if id exists, true if it not exists
    boolean found = false;

    if(this.users.isEmpty())
        found = true;
    else
    {
        for(int i=0;i<this.users.size();i++)
        {
            if(this.users.get(i).id == id)
            {
                found = false;
                break;
            }
            else
                found = true;
        }
    }
    return found;
}
```

context Person::actionPermission(action)
pre: self.accessibility -> exists (p: Person l p.action = Action) - - return true if the user action is found within the accessibility string

```java
//the below method checks if the action inserted by the user in the main class is permitted
@Override
//Refactoring
public boolean actionPermission(String action) {
    boolean actionFound = false;
    //the actions permitted are splitted into an array and then checked one by one with the action inserted by the user.
    String[] actions = this.accessibility.split("\n");
    for(String a:actions)
        if(a.contains(action))
        {
            actionFound = true;// if it has been found, it will return true
            break;
        }

    return actionFound;//else false, mening it is not permitted or does not exist.

}
```

context Person :: addUser (Person a)
pre: not self.id -> exists (p: Person | p.id = ID) - - The operation “addUser” will add new user if there is not an existing one

post: self.users -> exists (p: Person | p.users = Users) - - The pre-condition makes sure that the new user exists in the array

```
//the below method add a new user to the array of users
@Override
public void addUser(Person a){
    if(this.checkId(a.id) == true)//if the user id DOES NOT exists in the array
    {
        this.users.add(a);//add to the array
    }
}
```

Info.java

context Info :: checkDataId (id)
pre: self.database -> isEmpty = true - - return true if the info ID doesn’t exist in the database

post: self.database -> exists ((i: Info | i.database = Database) -> includes (i) and i.id = ID)) - - check the existence of the ID existing in the database

```
@Override
//OCL
//the fucntion below return false if the id is present in the array already, true otherwise
public boolean checkDataId(int id){
    boolean found = false;
    if(this.database.isEmpty())
        found = true;
    else
    {
        for(int i=0;i<this.database.size();i++)
        {
            if(this.database.get(i).id == id)
            {
                found = false;
                break;
            }
            else
                found = true;
        }
    }

    return found;
}
```

context Info :: printData ()
pre: self.database -> notEmpty (i: Info| i.dataID = ID and i.datacontent = content and i.userID = userID) - - the pre-condition checks if the data exist then displays the attribute of the Info details

post: self.database -> isEmpty (i:Info | i.database = Database) - - else the post-condition displays a string saying there is no existing data.

```
//the method below displays all the data present in the array (database)
@Override
public void printData(){
    if(!this.database.isEmpty())
    {
        this.database.forEach(a -> {
            System.out.println("\nData ID: "+a.id+"\nData content: "+a.content+"\nUser ID: "+a.userId);
        });
    }
    else
        System.out.println("\nNo data to be displayed.");
}
```

context Info :: editDataById (dataid, newcontent)
pre: self.database -> exists (i: Info | i.dataid) - - the pre-condition checks if dataid exists.

post: self.database -> exists (i: Info |i.dataid = id and i.newcontent = content)
- - then the post-condition checks the other info details

```
//the method below edits the content of the data with the id selected.
@Override
public void editDataById(int dataid, String newcontent){
    if(checkDataId(dataid))
    {
        this.database.get(dataIndexInArray(id)).content = newcontent;
        System.out.println("Changes have been made");
    }
    else
        System.out.println("Id does not exist. Try again.");
}
```

context Info :: deleteDataByID (dataid)
pre: self.database -> exists (i: Info| i.dataid = id) - - the pre-condition checks if the dataid exists
post: self.database -> database@pre -> reject (i: Info | i.dataid = id)
- - the post-condition removes the and updates the removal from the database

```
//the same method below first looks for the data to be deleted with the id, then invoke the method 'remove'
@Override
public void deleteDataById(int dataid){
    if(checkDataId(dataid))
    {
        this.database.remove(dataIndexInArray(id));
        System.out.println("Data deleted.");
    }
    else
        System.out.println("Id does not exist.");
}
```

Coursework3_Omar.java (Main class)

context Main :: generateIdUser()
pre: not person -> exists(p: Person | p.userId = id) - - the pre-condition check if the user id exist in the users array
post: person-> exists(p: Person | p.userId = id) - - the post-condition makes sure new user id exists

context Main :: generateIdData()
pre: not Info -> exists(i: Info | p.dataID = id) - - the pre-condition check for an data id
post: Info -> exists(i: Info | p.dataID = id) - - the post-condition makes sure new data id exists

```
public static int generateIdUser(){
    int userId = rd.nextInt(999);
    while(!users.checkId(userId))
        userId = rd.nextInt(999);
    return userId;
}

public static int generateIdData(){
    int dataId = rd.nextInt(999);
    while(!database.checkDataId(dataId))
        dataId = rd.nextInt(999);
    return dataId;
}
```

context Person :: loginUser ()
pre: not self.users -> exists (p:Person|p.userid = users) - - the pre-condition checks the user id doesn't exists
post: self.users -> exists (p:Person | p.userid = users) - - the post-condition checks the user exists

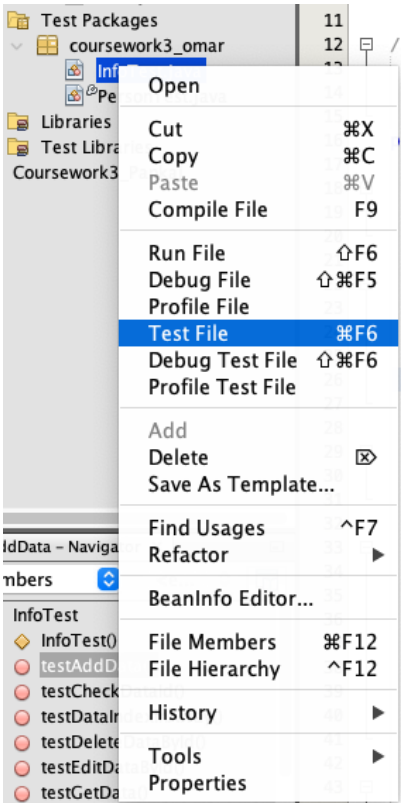
```
public static Person loginUser(){
    String userid;

    do{
        System.out.print("\nLOGIN\n\nInsert ID: ");
        userid = input.nextLine();
        try
        {
            System.out.println("User ID "+(users.checkId(Integer.parseInt(userid)) ? "does not exist. Try again or press 'B' to go back to
        }
        catch (NumberFormatException e)
        {
            break;
        }
    }while(users.checkId(Integer.parseInt(userid)));

    if("B".equals(userid))
        return null;
    else
        return users.getUser(Integer.parseInt(userid));
}
```

How to run code testing

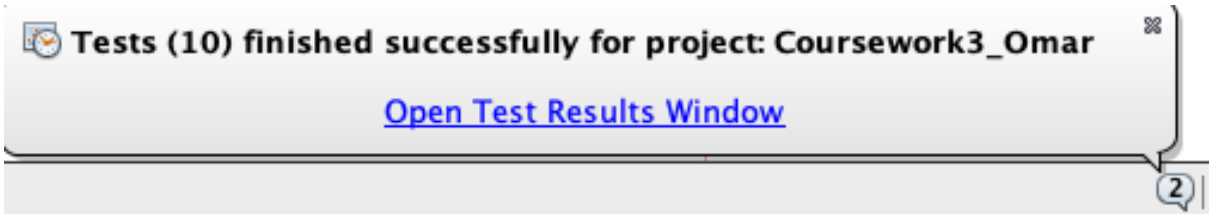
The code testing methods are situated inside the Test Packages folder, called "*InfoTest.java*" and "*PersonTest.java*".

The parameters of each methods have been written to test; therefore, they have no logical sense.

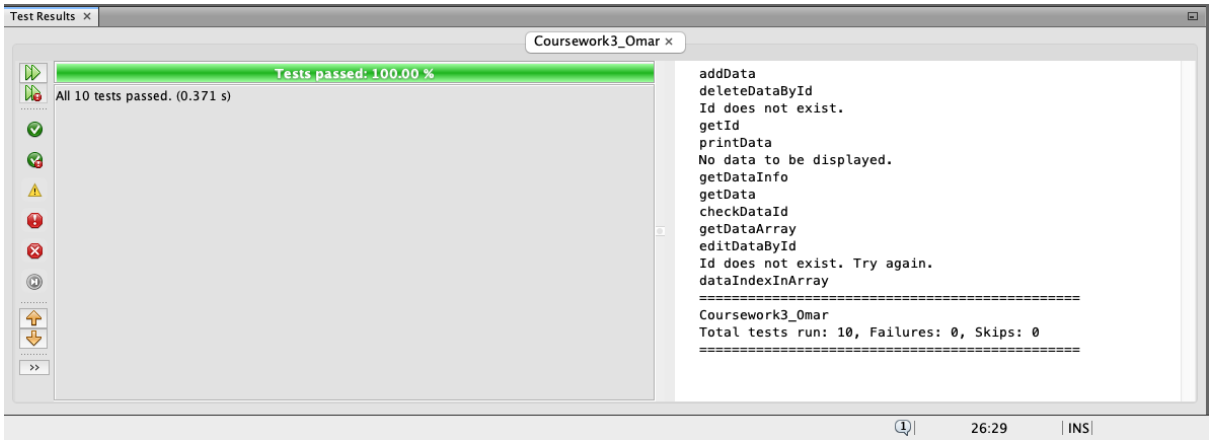
To run the test on NetBeans, the user must right-click on the file and select "Test File"



Then a pop up will appear from the bottom bar. The user must click on "Open Test Results Window"



NetBeans will then show you the methods that returns the right expected results. In this case all of them returned successfully with the pre-set result.



We can see in the right-side panel that some of the methods returns a string indicating that the method tested is not a function.

The example below shows how a test is running. The class tested is “Info.java” and the test method is situated in “InfoTest.java” inside the Test Packages folder.

```
/**
 * Test of checkDataId method, of class Info.
 */
@Test
public void testCheckDataId() {
    System.out.println("checkDataId");
    int id = 0;
    Info instance = new Info();
    boolean expResult = true;
    boolean result = instance.checkDataId(id);
    assertEquals(expResult, result);
}
```

In this case we initiate a variable which will be our test called ‘id’. The id will be checked inside an empty array of data created by the constructor in the class “Info”.

The method below returns false if the id is present in the array of data, true otherwise or if the array is empty.

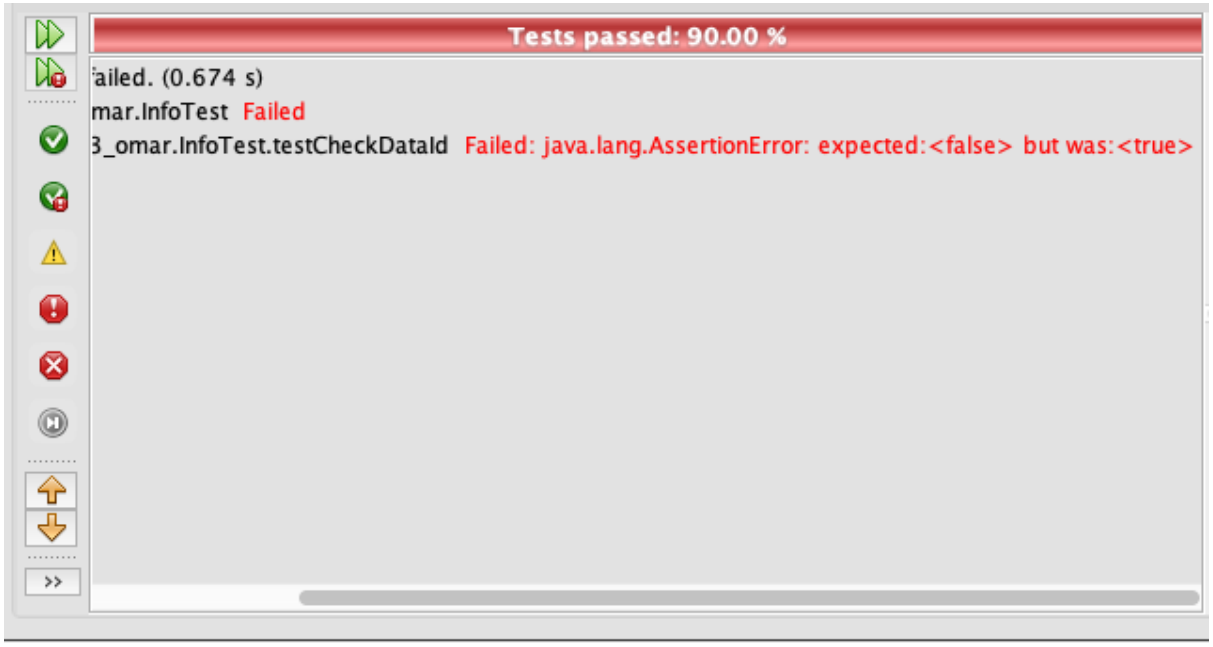
```
@Override
//OCL
//the fucntion below return false if the id is present in the array already, true otherwise
public boolean checkDataId(int id){
    boolean found = false;
    if(this.database.isEmpty())
        found = true;
    else
    {
        for(int i=0;i<this.database.size();i++)
        {
            if(this.database.get(i).id == id)
            {
                found = false;
                break;
            }
            else
                found = true;
        }
    }

    return found;
}
```

The expected result is “true”. The ‘id’ is equal to 0. The ArrayList of data is empty, so the method will stop at the first if condition returning “true”. Therefore, the test returns 0 error.

Now an example in case the expectation returns an error. I have changed the expectation variable to “false”.

```
/**
 * Test of checkDataId method, of class Info.
 */
@Test
public void testCheckDataId() {
    System.out.println("checkDataId");
    int id = 0;
    Info instance = new Info();
    boolean expResult = false;
    boolean result = instance.checkDataId(id);
    assertEquals(expResult, result);
}
```

Below the error displayed, underlining the expectation variable.



Design/implementation critique

The code included in this project has been built following the software development concepts for re-usability of components. I have not included a demonstration of an "1:m" association, because everything is carried out in the main class "*Coursework3_Omar.java*", which are present: code decomposition, code structure and code consistency. Although, a further improvement could be a class that manages the various instances reducing the amount of code in the main class, making it also clearer to read.

Overall, the module has been very clear and concise; it helped me planning and structure a software following a case study. I have also made the software code as simple and easy to read as possible. With the help of JUnit I have been able to test the methods by using the TDD approach and so to make the whole implementation functioning properly and with the expected results.

More examples of tests are included in the project folder under "*Test Packages*".

However, a lot of implementations regarding the code could have been added. The read-from-file approach is very slow but very efficient without an internet connection, but from the other side a database could be the right solution to speed up the process of writing and reading data from an external archive.

To save myself and the user some time, I have introduced the login by ID. Even though it should not be the right path to follow in terms of security, the log in method should have been replaced with a email and password check.

Without any doubt, a better graphic user interface (GUI) must be included in this project; the terminal interface could be confusing and, from my point of view, ugly for the user.

Another method that could be improved is the Shared Data. The user should only see the data shared with him. Also, the initial data displayed (user and data registered) are just to test the functionalities of the code. Therefore, the official software should not permit the user to know the other users details of course.