

ELECTRONIC ASSIGNMENT COVERSHEET



Murdoch
UNIVERSITY

Student Number	32783992
Surname	Camara
Given name	Mike
Email	mike@mikeios.com

Unit Code	ICT373
Unit name	Software Architectures
Enrolment mode	Internal
Date	16/April/2016
Assignment number	1
Assignment name	Assignment 1
Tutor	Robert

Student's Declaration:

- Except where indicated, the work I am submitting in this assignment is my own work and has not been submitted for assessment in another unit.
- This submission complies with Murdoch University's academic integrity commitments. I am aware that information about plagiarism and associated penalties can be found at <http://www.murdoch.edu.au/teach/plagiarism/>. If I have any doubts or queries about this, I am further aware that I can contact my Unit Coordinator prior to submitting the assignment.
- I acknowledge that the assessor of this assignment may, for the purpose of assessing this assignment:
 - reproduce this assignment and provide a copy to another academic staff member; and/or
 - submit a copy of this assignment to a plagiarism-checking service. This web-based service may retain a copy of this work for the sole purpose of subsequent plagiarism checking, but has a legal agreement with the University that it will not share or reproduce it in any form.
- I have retained a copy of this assignment.
- I will retain a copy of the notification of receipt of this assignment. If you have not received a receipt within three days, please check with your Unit Coordinator.

I am aware that I am making this declaration by submitting this document electronically and by using my Murdoch ID and password it is deemed equivalent to executing this declaration with my written signature.

Optional Comments to Tutor:

E.g. If this is a group assignment, list group members here

Mike Gomes Camaraz

*If you can, please insert this completed form into the body of **each** assignment you submit. Follow the instructions in the Unit Information and Learning Guide about how to submit your file(s) and how to name them, so the Unit Coordinator knows whose work it is.*

Table of contents

Question 1

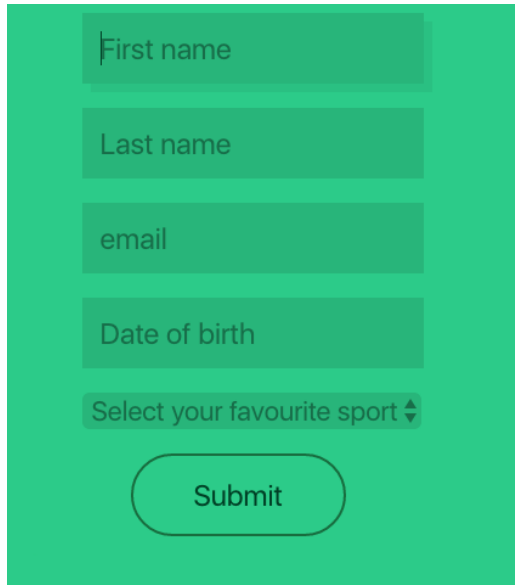
Description of the problem_____	3
My solution_____	3
Proof of testing and Sample results_____	4
Source code and Listing_____	5
FormValidation.html_____	5
ValidationForm.js_____	6
ValidationForm.css_____	9

Question 2

Title_____	11
Files_____	12
Statement of Purpose_____	12
Requirements/Specification_____	12
Structure/Design_____	12
UML _____	13
Limitations_____	14
Testing_____	14
Solution for questions A) to G)_____	16
Full code / Listing_____	22
Customer.java_____	22
Advertiser.java_____	31
Responder.java_____	41
DescPartnSought.java_____	45
Student.java_____	53
M8.java(main)_____	55

Description of the problem

The task requires the development of a form that collects a set of details about users and return this to a CGI destination to be tested. The required fields are first name, family name/surname, email address, date of birth, and favourite sport. Full validation is a must



My solution

I organised the form in three distinct files:

Formvalidation.html

validationForm.js

validationForm.css

The HTML page structure includes a CSS Stylesheet to enhance user interface design and a JavaScript script containing a function that test. The reason that I approached the problem with this modular perspective is that the code becomes more

maintainable. I ensured to include all required tags to make a HTML5 friendly and search engine optimised. The JavaScript validation function that prevents the user to leave empty fields or use inappropriate characters.

The CSS stylesheet file set attributes to the elements of the Page changing appearance and behaviour. My design approach was mobile first, and to use responsive elements like “em” or “%”.

Proof of testing and sample results

The figure consists of four screenshots of a web form titled "FORM" on a green background. Each screenshot shows a different validation error or successful submission state.

- Top Left:** The "First name" field is empty. A dialog box says: "This page says: Insert a valid name".
- Top Right:** The "First name" field contains "Mike" and the "Last name" field contains "Thomas". The "email" field contains "mu@". A dialog box says: "This page says: Enter a valid email address".
- Bottom Left:** The "First name" field contains "Mike" and the "Last name" field contains "Thomas". The "email" field contains "mu@email.com". A dialog box says: "This page says: Enter a valid date".
- Bottom Right:** The "First name" field contains "Mike" and the "Last name" field contains "Thomas". The "email" field contains "mu@email.com" and the "Date of birth" field contains "20/10/2016". A dialog box says: "This page says: Select your favourite sport from the list".

Each screenshot also shows a "Submit" button at the bottom.

ello ICT373 Student.

if CGI input variables were:

- [email] = [mu@email.com]
- [dob] = [20/10/2016]
- [name] = [Mike]
- [submit] = [Submit]
- [sport] = [Basketball]
- [surname] = [Thomas]

Source code listing

Formvalidation.html

```

1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <meta charset="utf-8">
5  <title>Registration Form</title>
6  <meta name="keywords" content="Form Validation, registration form, Software architecture" />
7  <meta name="description" content="This document is a Form Validation." />
8
9  <!-- Include css document to style page -->
10 <link rel="stylesheet" href="validationForm.css" type="text/css" />
11
12 <!-- Include JS document to validate form -->
13 <script src="validationForm.js"></script>
14 </head>
15
16 <!-- Start page body and focus on the name field -->
17 <body onload="document.registrationForm.name.focus();">
18   <div class="center-div">
19
20     <!-- Start form -->
21     <form method="POST" action="http://www.it.murdoch.edu.au/cgi-bin/reply1.pl" name="
registrationForm" onsubmit="return funcValidateForm(this)">
22       <ul>
23         <li><label>FORM</label></li>
24         <li><input type="text" name="name" placeholder="First name"/></li>
25         <li><input type="text" name="surname" placeholder="Last name" /></li>
26         <li><input type="text" name="email" placeholder="email"/></li>
27         <li><input type="text" name="dob" placeholder="Date of birth"/></li>
28         <li><select name="sport">
29
30           <option selected="" value="Default">Select your favourite sport</option>
31           <option value="Athletics">Athletics</option>
32           <option value="Basketball">Basketball</option>
33           <option value="Cricket">Cricket</option>
34           <option value="Football">Football</option>
35           <option value="Hockey">Hockey</option>
36           <option value="Swimming">Swimming</option>
37           <option value="Tennis">Tennis</option>
38         </select></li>
39         <li><input type="submit" name="submit" value="Submit" id="butSubmit"/></li>
40       </ul>
41     </form> <!-- End form -->
42   </div>
43 </body><!-- End body page -->
44
45 </html>

```

validationForm.js

```
1
2  /*
3     Mike Gomes Camara
4     ID: 32783992
5     Date: 12/April/2016
6     Creating a validation form in JS
7  */
8
9  function funcValidateForm()
10 {
11     var name = document.registrationForm.name;
12     var surname = document.registrationForm.surname;
13     var email = document.registrationForm.email;
14     var dob = document.registrationForm.dob;
15     var sport = document.registrationForm.sport;
16
17
18     if(onlyLetter(name))
19     {
20         if(onlyLetterSur(surname))
21         {
22             if(ValidateEmail(email))
23             {
24                 if(alphanumeric(dob))
25                 {
26                     if(selectSport(sport))
27                     {
28
29
30                     }
31                 }
32             }
33         }
34     }
35
36     return false;
37 }
38
39
40 /* Validate if field name contains only letters */
41 function onlyLetter(name)
42 {
43     var letters = /^[A-Za-z]+$/;
44     if(name.value.match(letters))
45     {
46         return true;
47     }
```

Cont...

validationForm.js

```

46     return true;
47 }
48 else
49 {
50     alert('Insert a valid name');
51     name.focus();
52     return false;
53 }
54 }
55
56 /* Validate if field surname contains only letters an numbers */
57 function onlyLetterSur(surname)
58 {
59     var letters = /^[0-9a-zA-Z]+$/;
60     if(surname.value.match(letters))
61     {
62         return true;
63     }
64     else
65     {
66         alert('Insert a valid surname');
67         surname.focus();
68         return false;
69     }
70 }
71
72 /* Validate if field email contains only letters, numbers,
73    and other characters that must have in a email field */
74 function ValidateEmail(email)
75 {
76     var mailformat = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
77     if(email.value.match(mailformat))
78     {
79         return true;
80     }
81     else
82     {
83         alert("Enter a valid email address");
84         email.focus();
85         return false;
86     }
87 }
88
89 /* Validate if field DOB contains only letters, numbers or the charcters "/" "." "-" */
90 function alphanumeric(dob)
91 {
92     var letters = /^[0-9a-zA-Z \\/ \. \- '_]+$/;

```

Cont...

validationForm.js

```
91 {
92   var letters = /^[0-9a-zA-Z \\/ \. \-'\_]+$/;
93   if(dob.value.match(letters))
94   {
95     return true;
96   }
97   else
98   {
99     alert('Enter a valid date');
100    dob.focus();
101    return false;
102  }
103 }
104
105 /* Validate if field sport contains is selected or different of default */
106 function selectSport(sport)
107 {
108   if(sport.value == "Default")
109   {
110     alert('Select your favourite sport from the list');
111     sport.focus();
112     return false;
113   }
114   else
115   {
116     alert('Form submitted');
117     window.open(www.it.murdoch.edu.au/cgi-bin/reply1.pl);
118     window.location.reload();
119   }
120   return true;
121 }
122 }
123
```


validationForm.css

```
2  /*
3     Mike Gomes Camara
4     ID: 32783992
5     Date: 12/April/2016
6     Style CSS of my form
7  */
8
9  /* All characteres become thinner */
10 html {
11     -webkit-font-smoothing: antialiased;
12 }
13
14 /* Change font style of label */
15 label {
16     font-family: "HelveticaNeue-Light", "Helvetica Neue Light", "Helvetica Neue", Helvetica, Arial, "Lucida
17     color: #19724d;
18     width: 100%;
19     font-size: 1.5em;
20 }
21
22 /* Remove dot list from form */
23 form li {
24     list-style: none;
25     margin-bottom: 10px;
26 }
27
28 /* Change the entire page color to green */
29 body {
30     background-color: #2ccb89;
31 }
32
33 /* Change the placeholder text color */
34 ::-webkit-input-placeholder {
35     color: #186d49;
36 }
37
38 :-moz-placeholder { /* Firefox 18- */
39     color: #186d49;
40 }
41
42 ::-moz-placeholder { /* Firefox 19+ */
43     color: #186d49;
44 }
45
46 :-ms-input-placeholder {
47     color: #186d49;
```

validationForm.css

```
47     color: #000080;
48 }
49
50 /* Style DIV that holds the form in the page document */
51 .center-div
52 {
53     margin: 5%;
54     width: 100%;
55     height: 100%;
56     border-radius: 3px;
57     background-color: #2ccb89;
58 }
59
60
61 /* Add shadow when user click on input fields */
62 input:focus {
63     outline: none !important;
64     box-shadow: 7px 7px 0px #2ac082;
65 }
66
67 /* Style input text field */
68 input[type=text] {
69     padding: 15px;
70     margin-bottom: 10px;
71     border: 0px none #ccc;
72     font-size: 1.5em;
73     background-color: #28b57a;
74     color: #003f25;
75 }
76
77 /* Style submit button */
78 input[type="submit"] {
79     padding: 18px 50px;
80     cursor: pointer;
81     -webkit-border-radius: 40px;
82     border-radius: 40px;
83     border-color: rgba(24, 109, 63, 1.0);
84     background-color: rgba(44, 203, 137, 0.0);
85     border-style: solid;
86     font-size: 1.5em;
87     color: #003f25;
88     margin-left: 40px;
89     margin-top: 10px;
90 }
91
92 /* Change color of submit button when hover */
93 input[id=butSubmit]:hover{
```

```
/* Add style for the sports input field */
select{
  border: none;
  box-shadow: none;
  background-color: #28b57a;
  background-image: none;
  font-size: 1.4em;
  color: #186d49;
}

/* Add shadow when user click on sports |input fields */
select:focus {
  outline: none !important;
  box-shadow: 7px 7px 0px #2ac082;
}
```



All the files above are zipped in the file form.zip also attached in the project.

Question 2

Design, implement (in Java), test and document a set of classes for use in a program to manage a (prototype) online dating service.



Title M8 ® (mate) Dating App , @author - Mike Gomes

Files

Advertiser.java

Customer.Java

DescPartnSought.java

M8.java (main program)

Responder.java

Student.java

Statement of purpose

Industry disruptor matching partners algorithm based on their characteristics and preferences.

Requirements/Specification

To be able to run my M8 app the user would need to have java compiler installed in their computer.

The user Run the file from the M8.java, all required inputs were hardcoded so no keyboard input is required to fully test the program.

The output file is then prompted in the user console.

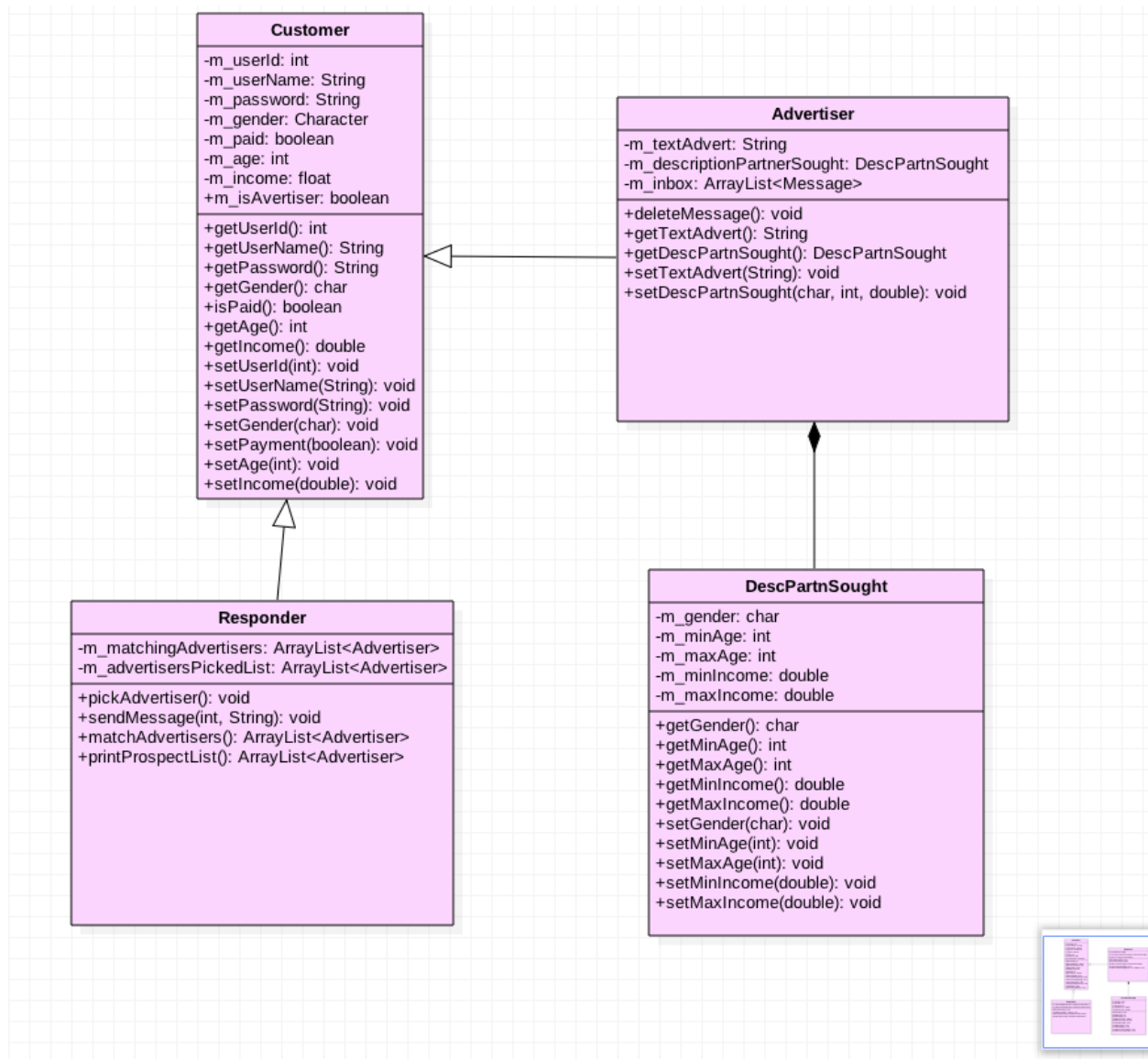
Structure/Design

I've chosen to approach the software architecture of this program using an object oriented design strategy that aims to keep the code easy to maintaining and to escalate up.

The structure includes Inheritance relations like Advertiser that is a subclass of Customer which is also parent of Responder, in a way to avoid redundancy with high cohesion.

I decided to approach this modular concept because if in the future we decide to increase the filters of partner sought or add attributes to one of the classes the changes keep focused on the scope without having to change elsewhere.

UML



Limitations

The code does not implement graphic interface, all commands and results are given using console.

Testing

Testing is where we developers spend most of our time, and definitely spend more time debugging then creating logic. Constant iterations of testing reduces debugging time. That is why I prefer to understand deeply the code while it is running fine and as I work on the algorithm test very often, and comment effectively. Although, I haven't take this approach for this particular assignment, It is also important to create testing classes that test the methods and variables of the class systematically.

Printing logs in the console is also very effective to track down errors, that is why you can understand what is going on my code by looking the outputs.

Age: 32
Text advert: I am Nati

Sending to messages from Luke to Natalia
#####

Logging in Avertiser....
#####

Hi Natalia

INBOX

You have 2 messages

[Luke said - I love you so much, Luke said - Lets get married]
[Luke said - I love you so much, Luke said - Lets get married]
#####

Deleting message 1...
Printing new INBOX...

#####

Hi Natalia

INBOX

You have 1 messages

Listing

List all my code, but first comment accord java doc

The client program should complete few tasks, so I listed them below individually to facilitate assessment but I will fully list the code afterwards.

a) create a list of 6-7 different customers of both types with made-up details built in to the client program

```
System.out.println("Added 4 new Advertisers and their partner sought" + "\n");
// Create 4 new advertisers
Advertiser adv1 = new Advertiser(123, "Marina", "test", 'F', true, 32,120000, "I am Mima", inbox);
Advertiser adv2 = new Advertiser(1234, "Natalia", "test", 'F', true, 32,120000, "I am Nati", inbox);
Advertiser adv3 = new Advertiser(12345, "Nicole", "test", 'F', true, 32,120000, "I am Nici", inbox);
Advertiser adv4 = new Advertiser(123456, "Sheila", "test", 'F', true, 32,120000, "I am Sheilinha", inbox);

// Add advertisers to a list
listAdvertisers.add(adv1);
listAdvertisers.add(adv2);
listAdvertisers.add(adv3);
listAdvertisers.add(adv4);

// Add Description Partner Sought for each advertiser
adv1.m_desiredPartner.setDesiredPartner('M', 23, 37, 20000, 130000);
adv2.m_desiredPartner.setDesiredPartner('M', 32, 37, 20000, 130000);
adv3.m_desiredPartner.setDesiredPartner('M', 40, 70, 20000, 100000);
adv4.m_desiredPartner.setDesiredPartner('M', 70, 80, 20000, 100000);

System.out.println("\n");

System.out.println("Added 4 new Responders" + "\n");
// Add 4 new responders
Responder res1 = new Responder(113, "Luke", "test", 'M', true, 32, 120000);
Responder res2 = new Responder(114, "John", "test", 'M', true, 30, 120000);
Responder res3 = new Responder(112, "Mark", "test", 'M', true, 28, 120000);
Responder res4 = new Responder(115, "Mathew", "test", 'M', true, 25, 120000);

// Add responders to a list
listResponders.add(res1);
```

```
Output - m8 (run) X
```

```

Added 4 new Responders

List of all advertisers:

Marina
Natalia
Nicole
Sheila

List of all responders:

Luke
John
Mark
Mathew

```


b) get some matches for a responder, choose one match and send the match a message, then log in that advertiser to get the message

```
// Start process of sending message

// Get Responder name
String resName = res1.getUserName();

// Get responder input for choosing partner
int chosenPartner = 1;

// Get input for text message
String loveMessage = "I love you so much";
String loveMessage2 = "Lets get married";

System.out.println( "\n");
System.out.println("Sending to messages from " + resName + " to " + listMatched.get(1).getUserName());

System.out.println("#####");
// Send messages to partner chosen
listMatched.get(chosenPartner).addMessage(resName, loveMessage);
listMatched.get(chosenPartner).addMessage(resName, loveMessage2);

// Advertiser screen after login

System.out.println( "\n");
System.out.println("Logging in Avertiser...");

System.out.println("#####");
System.out.println("\n" + "Hi " + listMatched.get(1).getUserName() + "\n" );

System.out.println("##### INBOX #####");
```

```

Output - m8 (run) x
Logging in Responder....
#####
Hello Luke

You have 2 matches

List of your matches

##### Match 0 #####
0 - Marina
Age: 32
Text advert: I am Mima

##### Match 1 #####
1 - Natalia
Age: 32
Text advert: I am Nati

Sending to messages from Luke to Natalia
#####

Logging in Avertiser....
#####

Hi Natalia

##### INBOX #####

You have 2 messages

[Luke said - I love you so much, Luke said - Lets get married]
[Luke said - I love you so much, Luke said - Lets get married]

```

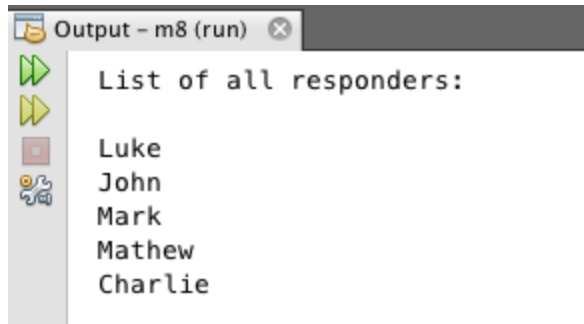
c) add a new customer to the dating service

```

// Add new cutomer to the system
// Object type Responder, with ID, username, gender, isPaid, age, income
Responder res5 = new Responder(115, "Charlie", "test", 'M', true, 25, 120000);

// Add responders to a list
listResponders.add(res5);

```



d) delete an existing customer from the dating service

```

//Remove customer process

// Input user id to delete
int idToDelete = 113;

// Delete chosen ID
for (Iterator<Responder> it = listResponders.iterator(); it.hasNext(); )
{
    Responder res = it.next();
    if (res.getUserId() == idToDelete)
    {
        it.remove();
        System.out.println("Deleting responder " + res.getUserName() + "...");
    }
}

// End delete user

System.out.println("\n" + "List of all responders after delete one:" + "\n");

//List all Responders
for (int i=0; i<listResponders.size();i++)
{
    String value = listResponders.get(i).getUserName();
    System.out.printf(value + "\n");
}
System.out.println("\n");
  
```

```
#####
Deleting responder Luke...
```

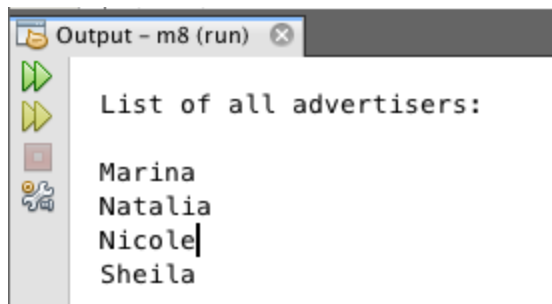
List of all responders after delete one:

```
John
Mark
Mathew
Charlie
```

e) display the details of all advertisers registered with the service

```
// List all Advertisers
currentUserAdv.printListAdvertisers(listAdvertisers);

public void printListAdvertisers(ArrayList<Advertiser> listAdv)
{
    for (int i=0; i<listAdv.size();i++)
    {
        String value = listAdv.get(i).getUser_name();
        System.out.printf(value + "\n");
    }
}
```



f) display the details of all responders registered with the service

```
// List all Responders
currentUserRes.printListResponders(listResponders);
```

```

public void printListResponders(ArrayList<Responder> listRes)
{
    for (int i=0; i<listRes.size();i++)
    {
        String value = listRes.get(i).getUserName();
        System.out.printf(value + "\n");
    }
}

```

List of all responders:

Luke
John
Mark
Mathew
Charlie

g) repeat steps (b) to (d) above for different advertisers and responders to thoroughly test your program.

The left screenshot shows the following output:

```

System is login in...
Mark is loggedIn
b) get some matches for a responder...

Logging in Responder....
#####
Hello John

You have 4 matches

List of your matches

##### Match 0 #####
0 - Marina
Age: 32
Text advert: I am Mima

```

The right screenshot shows the following output:

```

You have 1 messages

[Luke said - I love you so much]
#####
d) delete an existing customer from the dating service

Deleting responder Luke..

List of all responders after delete one:

John
Mark
Mathew
Charlie

BUILD SUCCESSFUL (total time: 0 seconds)

```

Listing

Full code of M8 Dating System

Customer.java

```
// ICT373

// Assignement 1

// Task: Develop a console operated dating system

// This class is the main where we test the program

// M8 stands for Mate is the Name of the app.


/**
 * @author   Mike Gomes <mike@mikeios.com>
 * @version  1.0          (current version number of program)
 * @since    12-April-2016  (the version of the package this class was first added to)
 */

package m8;


/**
 *
 * Class customer is the parent of the classes Advertiser and Customer
 */

public class Customer {
```

```
/**  
 * user id, unique identifier for users  
 */  
private int m_userId;  
  
/**  
 * username, how user would like to be called  
 */  
private String m_userName;  
  
/**  
 * user password  
 */  
private String m_password;  
  
/**  
 * gender of the user  
 */  
private char m_gender;
```

```
/**
 * boolean variable that is true if user has paid their fees
 */
private boolean m_paid;

/**
 * user age
 */
private int m_age;

/**
 * user income
 */
private double m_income;

/**
 * six-argument constructor
 */
public Customer(int userId, String userName, String password, char gender,
                boolean paid, int age, double income)
{
```



```
this.m_userId = userId;

this.m_userName = userName;

this.m_password = password;

this.m_gender = gender;

this.m_paid = paid;

this.m_age = age;

this.m_income = income;

} // end constructor


/**
 * get user id
 */
public int getUserId()
{
    return m_userId;
}


/**
 * get username
 * @return username
 */
```

```
public String getUserName()
{
    return m_userName;
}
```

```
/**
```

```
 * get password
```

```
 * @return password
```

```
 */
```

```
public String getPassword()
{
    return m_password;
}
```

```
/**
```

```
 * get gender
```

```
 * @return gender
```

```
 */
```

```
public char getGender()
{
    return m_gender;
}
```

```
}
```

```
/**
```

```
 * Boolean that is true if user paid fees
```

```
 * @return isPaid
```

```
 */
```

```
public boolean isPaid()
```

```
{
```

```
    return m_paid;
```

```
}
```

```
/**
```

```
 * get user age
```

```
 * @return age
```

```
 */
```

```
public int getAge()
```

```
{
```

```
    return m_age;
```

```
}
```

```
/**
```

```
* get user income
* @return income
*/

public double getIncome()
{
    return m_income;
}

/**
 * set user id
 * @param userId
 */
public void setUserId(int userId)
{
    this.m_userId = userId;
}

/**
 * set username
 * @param username
 */
```

```
public void setUsername(String username)

{

    this.m_username = username;

}
```

```
/**
```

```
 * set user password
```

```
 * @param password
```

```
 */
```

```
public void setPassword(String password)
```

```
{
```

```
    this.m_password = password;
```

```
}
```

```
/**
```

```
 * set user gender
```

```
 * @param gender
```

```
 */
```

```
public void setGender(char gender)
```

```
{
```

```
    this.m_gender = gender;
```

```
}
```

```
/**
```

```
 * set boolean true is user paid fees
```

```
 * @param paid
```

```
 */
```

```
public void setIsPaid(boolean paid)
```

```
{
```

```
    this.m_paid = paid;
```

```
}
```

```
/**
```

```
 * set user age
```

```
 * @param age
```

```
 */
```

```
public void setAge(int age)
```

```
{
```

```
    this.m_age = age;
```

```
}
```

```
/**
```

```

    * set user income

    * @param income

    */

    public void setIncome(double income)

    {

        this.m_income = income;

    }

} //end of class Customer

```

Advertiser.java

```

// ICT373

// Assignement 1

// Task: Develop a console operated dating system

// This class is the main where we test the program

// M8 stands for Mate is the Name of the app.


/**

 * @author   Mike Gomes <mike@mikeios.com>

 * @version  1.0          (current version number of program)

 * @since    12-April-2016  (the version of the package this class was first added to)

 */

package m8;

```

```
import java.util.ArrayList;

import java.util.Iterator;


/**
 *
 * Class Advertiser inherits from Customer
 * so it extends the methods form Customer like getName()
 */
public class Advertiser extends Customer{


    /**
     * String to hold text advertising
     */
    private String m_textAdvert;


    /**
     * A message is a String
     */
    private String m_message;


    /**
```


* This holds an array of Strings that are the messages from the Responders

* in the future think about creating a class Message

*/

```
ArrayList<String> m_inbox = new ArrayList<String>();
```

/**

* Creates an object of the Type DescPartnSought

* that will be set according to the preferences of the Advertiser

*/

```
DescPartnSought m_desiredPartner = new DescPartnSought();
```

/**

* 7 argument constructor

* (subclass) constructor that calls the superclass constructor and then adds initialization code of its own

*/

```
public Advertiser(int userId, String userName, String password, char gender,
    boolean paid, int age, double income, String textAdvert, ArrayList<String> inbox)
```

```
{
```

```
    // Call the super class Customer which holds those private variables
```

```
    super(userId, userName, password, gender, paid, age, income);
```

```
        this.m_textAdvert = textAdvert;

        this.m_inbox = inbox;
    }

    /**
     * Get text advert method returns a text advert
     */
    public String getTextAdvert()
    {
        return m_textAdvert;
    }

    /**
     * set text advert method
     */
    public void setTextAdvert(String textAdvert)
    {
        this.m_textAdvert = textAdvert;
    }
}
```

```
/**  
 * Set desired gender  
 */  
  
public void setDesiredGender(char gender)  
{  
    m_desiredPartner.setDesiredGender(gender);  
}
```

```
/**  
 * Set desired minimum age  
 */  
  
public void setDesiredMinAge(int age)  
{  
    m_desiredPartner.setDesiredMinAge(age);  
}
```

```
/**  
 * Set desired max age  
 */  
  
public void setDesiredMaxAge(int age)  
{
```

```
m_desiredPartner.setDesiredMaxAge(age);  
  
}  
  
/**  
 * set desired min income  
 */  
public void setDesiredMinIncome(double minIncome)  
{  
    m_desiredPartner.setDesiredMinIncome(minIncome);  
}  
  
/**  
 * set desired max income  
 */  
public void setDesiredMaxIncome(double maxIncome)  
{  
    m_desiredPartner.setDesiredMaxIncome(maxIncome);  
}  
  
/**  
 * Get desired gender
```

```
*/  
  
public char getDesiredGender()  
  
{  
  
    return m_desiredPartner.getDesiredGender();  
  
}  
  
  
/**  
  
 * Get desired minimum age  
  
 */  
  
public int getDesiredMinAge()  
  
{  
  
    return m_desiredPartner.getDesiredMinAge();  
  
}  
  
  
/**  
  
 * Get desired max age  
  
 */  
  
public int getDeridedMaxAge()  
  
{  
  
    return m_desiredPartner.getDesiredMaxAge();  
  
}
```

```
/**
 * Get desired min income
 */
public double getDesiredMinIncome()
{
    return m_desiredPartner.getDesiredMinIncome();
}

/**
 * Get desired max income
 */
public double getDesiredMaxIncome()
{
    return m_desiredPartner.getDesiredMaxIncome();
}

/**
 * Method to add messages to the Advertiser Inbox(ArrayList of strings)
 * it have a Responder name(owner) and a message
 * @param responder
```

```
* @param message
*
*/

public void addMessage(String resName, String message)
{
    String msg = resName + " said - " + message;
    m_inbox.add(msg);
}

/**
 * This method deletes a message of an Inbox taking a number as a parameter
 * @param index
 */

public void deleteMessage(int index)
{
    m_inbox.remove(index);
}

/**
 * Get message
```

```
* @return message
*/

public String getMessage()
{
    return m_message;
}

/**
 * Loops arrayList of messages and print them on the console
 */
public void printInbox()
{
    for (int i=0; i<m_inbox.size();i++)
    {

        String value = m_inbox.toString();

        System.out.printf(value + "\n");

    }
}

/**
```


* Takes a list of Advertiser as a parameter, loop through and print all on console

* @param Advertiser

*/

```
public void printListAdvertisers(ArrayList<Advertiser> listAdv)
```

```
{
```

```
    for (int i=0; i<listAdv.size();i++)
```

```
    {
```

```
        String value = listAdv.get(i).getUserName();
```

```
        System.out.printf(value + "\n");
```

```
    }
```

```
}
```

```
// end class Advertiser
```

Responder.java

```
// ICT373
```

```
// Assignment 1
```

```
// Task: Develop a console operated dating system
```

```
// This class is the main where we test the program
```

```
// M8 stands for Mate is the Name of the app.
```

```
/**
```

```
* @author Mike Gomes <mike@mikeios.com>
```

```
* @version 1.0 (current version number of program)
* @since 12-April-2016 (the version of the package this class was first added to)
*/

package m8;

import java.util.ArrayList;
import java.util.Iterator;

/**
 *
 * Class Responder inherits from Customer
 * so it extends the methods form Customer like getName()
 */
public class Responder extends Customer {

    /**
     * String to hold text message
     */
    private String m_msg;

    /**
```

* int that holds a number that represents the Advertiser that the responder

* will send a message

*/

private int m_favMatch;

/**

* Holds state of user in the system if true so user is logged in

*/

private boolean m_isLoggedIn;

/**

* 7 argument constructor

* (subclass) constructor that calls the superclass constructor and then adds initialization code of its own

*/

public Responder(int userId, String userName, String password, char gender,

boolean paid, int age, double income)

{

// Call the super class Customer which holds those private variables

super(userId, userName, password, gender, paid, age, income);

```
}
```

```
/**
```

```
 * Method to send message to Advertisers
```

```
 * @param number that holds advertiser and string that holds a message
```

```
 */
```

```
public void sendMessage(int favMatch, String msg)
```

```
{
```

```
    m_favMatch = favMatch;
```

```
    m_msg = msg;
```

```
}
```

```
/**
```

```
 * Method to print list of Responder
```

```
 * @param ArrayList of Responder
```

```
 */
```

```
public void printListResponders(ArrayList<Responder> listRes)
```

```
{
```

```
    for (int i=0; i<listRes.size();i++)
```

```
    {
```

```

        String value = listRes.get(i).getUserName();

        System.out.printf(value + "\n");

    }

}

/**
 * set state of user in the system, if true user is logged in
 */
public void setIsLoggedIn(boolean isLoggedIn)
{
    m_isLoggedIn = isLoggedIn;
}

} // end of Responder class

```

DescPartnSought.java

```

// ICT373

// Assignement 1

// Task: Develop a console operated dating system

// This class is the main where we test the program

// M8 stands for Mate is the Name of the app.

```

```
/**  
  
 * @author   Mike Gomes <mike@mikeios.com>  
  
 * @version  1.0          (current version number of program)  
  
 * @since    12-April-2016  (the version of the package this class was first added to)  
  
 */  
  
package m8;  
  
  
/**  
  
 *  
  
 * @author mcmikecamara  
  
 */  
  
 *  
  
 * Class DescPartnSought is a class that will be used as an object of the Advertiser  
  
 * and will hold all the requirements of the advertiser for their choice of soulmate  
  
 */  
  
public class DescPartnSought {  
  
  
    /**  
  
     * desired partner gender  
  
     */  
  
    private char m_desiredGender;
```

```
/**
```

```
 * desired partner minimum age
```

```
 */
```

```
private int m_desiredMinAge;
```

```
/**
```

```
 * desired partner Maximum age
```

```
 */
```

```
private int m_desiredMaxAge;
```

```
/**
```

```
 * desired partner minimum income
```

```
 */
```

```
private double m_desiredMinIncome;
```

```
/**
```

```
 * desired partner maximum income
```

```
 */
```

```
private double m_desiredMaxIncome;
```

```
/**
 * Constructors that take no parameters
 */
public DescPartnSought(){}

/**
 * Get desired partner gender
 * @return desiredGender
 */
public char getDesiredGender()
{
    return m_desiredGender;
}

/**
 * Set desired partner gender
 * @param gender
 */
public void setDesiredGender(char gender)
{
    m_desiredGender = gender;
}
```



```
}
```

```
/**
```

```
 * Get desired partner minimum age
```

```
 * @return desired minimum age
```

```
 */
```

```
public int getDesiredMinAge()
```

```
{
```

```
    return m_desiredMinAge;
```

```
}
```

```
/**
```

```
 * Set desired partner min age
```

```
 * @param minAge
```

```
 */
```

```
public void setDesiredMinAge(int age)
```

```
{
```

```
    m_desiredMinAge = age;
```

```
}
```

```
/**
```

```
* Get desired partner maximum age
* @return desired partner maximum age
*/

public int getDesiredMaxAge()
{
    return m_desiredMaxAge;
}

/**
 * Set desired partner max age
 * @param maxAge
 */
public void setDesiredMaxAge(int age)
{
    m_desiredMaxAge = age;
}

/**
 * Get desired partner income
 * @return partner income
 */
public double getDesiredMinIncome()
```

```
{  
    return m_desiredMinIncome;  
}  
  
/**  
 * Set desired partner min income  
 * @param minIncome  
 */  
public void setDesiredMinIncome(double minIncome)  
{  
    m_desiredMinIncome = minIncome;  
}  
  
/**  
 * Get desired partner max income  
 * @return maximum income  
 */  
public double getDesiredMaxIncome()  
{  
    return m_desiredMaxIncome;  
}
```

```
/**
 * Set desired partner income
 * @param maxIncome
 */
public void setDesiredMaxIncome(double maxIncome)
{
    m_desiredMaxIncome = maxIncome;
}

/**
 * Set Desired partner
 * @param gender, minAge, maxAge, minIncome, maxIncome
 */
public void setDesiredPartner(char gender, int minAge, int maxAge, double minIncome, double
maxIncome)
{
    m_desiredGender= gender;
    m_desiredMinAge = minAge;
    m_desiredMaxAge = maxAge;
    m_desiredMinIncome = minIncome;
    m_desiredMaxIncome = maxIncome;
}
```

```
}
```

```
// End class DescPartnSought
```

Student.java

```
// ICT373
```

```
// Assignement 1
```

```
// Task: Develop a console operated dating system
```

```
// This class is the main where we test the program
```

```
// M8 stands for Mate is the Name of the app.
```

```
/**
```

```
 * @author   Mike Gomes <mike@mikeios.com>
```

```
 * @version  1.0          (current version number of program)
```

```
 * @since    12-April-2016  (the version of the package this class was first added to)
```

```
 */
```

```
package m8;
```

```
// Libraries imported to be able to print current date on console
```

```
import java.text.DateFormat;
```

```
import java.text.SimpleDateFormat;
```

```
import java.util.Calendar;
```

```
/**
```

*

* Class Student is used uniquely to print my details as a student on the main program

*/

```
public class Student {
```

```
    /**
```

```
    *
```

```
    * Get current date time with Calendar()
```

```
    */
```

```
    DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm");
```

```
    Calendar cal = Calendar.getInstance();
```

```
    /**
```

```
    *
```

```
    * Method to print my details as a student and date on the console
```

```
    */
```

```
    public void printStudentDetails()
```

```
    {
```

```
        System.out.println("Student: Mike Gomes Camara" + "\n" + "ID: 32783992" + "\n" +
```

```
            "Mode enrolment: Internal" + "\n" + "Tutor names: Ferdous and Rob" + "\n" +
```

```
            "Today is: " + dateFormat.format(cal.getTime()) + "\n" );
```

```
}
```

```
} // End class Student
```

M8.java (main)

```
// ICT373
```

```
// Assignement 1
```

```
// Task: Develop a console operated dating system
```

```
// This class is the main where we test the program
```

```
// M8 stands for Mate is the Name of the app.
```

```
/**
```

```
 * @author    Mike Gomes <mike@mikeios.com>
```

```
 * @version   1.0          (current version number of program)
```

```
 * @since     12-April-2016    (the version of the package this class was first added to)
```

```
 */
```

```
package m8; //Included in the package m8
```

```
// import library to provide arraylist functionality
```

```
import java.util.ArrayList;
```

```
import java.util.Iterator;
```

```
/**
```

```
* Tests a matching partners algorithm.
```

```
*
```

```
* To try access with different users just uncomment the lines in the main.
```

```
*
```

```
*
```

```
*/
```

```
public class M8 {
```

```
/**
```

```
* Main method, where we run the program (1)
```

```
* <p>
```

```
* Attention! Important. Read the comments and uncomment lines to different outputs. [2]
```

```
*
```

```
* <p>
```

```
* The program does not take keyboard input.
```

```
*
```

```
* The program addresses all requirements of the tasks including those below.
```

```
*
```

```
* The service manages a list of customers.
```

```
* Customers have login (names) and passwords.
```


- * A customer is either an advertiser or a responder.
 - * All customers also have some personal details including gender, age and income.
 - * An advertiser has a text advert, a description of partner sought and a list of reply messages.
 - * A description of partner sought has a gender, range of ages and range of incomes.
 - * A reply message has an owner (which is a responder) and some text.
- */

```
public static void main(String[] args) {
```

```
    /**
```

```
    * Object of Type Student which will be used to print my detail on console
```

```
    */
```

```
    Student student = new Student();
```

```
    /**
```

```
    * ArrayList of strings that will hold love messages from the Responders
```

```
    */
```

```
    ArrayList<String> inbox = new ArrayList<String>();
```

```
    /**
```

```
    * ArrayList that holds a list of objects of the type Advertiser
```

```
*/  
  
ArrayList<Advertiser> listAdvertisers = new ArrayList<Advertiser>();  
  
/**  
 * ArrayList that holds a list of Responders  
 */  
  
ArrayList<Responder> listResponders = new ArrayList<Responder>();  
  
/**  
 * ArrayList that holds a list of objects of the type Advertiser  
 * in this case the advertisers which will be displayed to the Responder  
 * in case of matching criteria  
 */  
  
ArrayList<Advertiser> listMatched = new ArrayList<Advertiser>();  
ArrayList<Advertiser> listMatched2 = new ArrayList<Advertiser>();  
  
/**  
 * Boolean variable that represents the state of a user in the system,  
 * where loggedIn false means that the user is not on the system  
 */  
  
boolean loggedIn = false;
```

```

/**
 * Created a object of Advertiser that will hold the current user
 */
Advertiser currentUserAdv = new Advertiser(1, "", "", 'F', false, 1,1, "", inbox);

/**
 * Created a object of Advertiser that will hold the current user
 */
Responder currentUserRes = new Responder(1, "", "", 'M', false, 1, 1);

/**
 * Method that prints my student detail in the console using a student object
 */
student.printStudentDetails();

/**
 * print a user interface on the console
 */
System.out.println("#####" +
"\n");

System.out.println("###      M8 Dating System      ##### + "\n");

```

```
System.out.println("#####" +
"\n");
```

```
System.out.println("\n");
```

```
System.out.println("Adding 4 new Advertisers and their partner sought.." + "\n");
```

```
System.out.println("Done!" + "\n");
```

```
/**
```

```
* Required. The service manages a list of customers.
```

```
*/
```

```
System.out.println("a) create a list of 6-7 different customers of both types with made-up
details built in to the client\n");
```

```
/**
```

```
*
```

```
* Required. An advertiser has a text advert, a description of partner sought and a list of
reply messages.
```

```
* Required. An advertiser has a text advert, a description of partner sought and a list of
reply messages.
```

```
* Required. Customers have login (names) and passwords.
```

```
* When a customer is created in this the advertiser, the constructor
```

```
* method assign those variables- ID, username, password,gender,paid,age, income, text
advert, an empty inbox
```

```
*/
```

```
Advertiser adv1 = new Advertiser(123, "Marina", "test", 'F' , true, 32,120000, "I am Mima"  
, inbox);
```

```
Advertiser adv2 = new Advertiser(1234, "Natalia","test", 'F', true, 32,120000, "I am Nati",  
inbox);
```

```
Advertiser adv3 = new Advertiser(12345, "Nicole", "test", 'F', true, 32,120000, "I am Nici",  
inbox);
```

```
Advertiser adv4 = new Advertiser(123456, "Sheila", "test", 'F', true, 32,120000, "I am  
Sheilinha", inbox);
```

```
/**
```

```
 * Add advertisers to a list of all advertisers in the system
```

```
 */
```

```
listAdvertisers.add(adv1);
```

```
listAdvertisers.add(adv2);
```

```
listAdvertisers.add(adv3);
```

```
listAdvertisers.add(adv4);
```

```
/**
```

```
 * Required. A description of partner sought has a gender, range of ages and range of  
incomes.
```

```
 * Every advertiser has a private object of type Description Sought
```

```
 * that has a method to set description of the desired match
```

```
 * The constructor object have 3 criteria, but in future it can be updated
```

* Add Description Partner Sought have-Gender,min age, max age, min income , max income

*/

adv1.m_desiredPartner.setDesiredPartner('M', 23, 37, 50000, 140000);

adv2.m_desiredPartner.setDesiredPartner('M', 32, 37, 50000, 150000);

adv3.m_desiredPartner.setDesiredPartner('M', 30, 70, 50000, 160000);

adv4.m_desiredPartner.setDesiredPartner('M', 30, 80, 50000, 170000);

System.out.println("\n");

System.out.println("Adding 4 new Responders.." + "\n");

System.out.println("Done!" + "\n");

/**

* Required. A customer is either an advertiser or a responder.

* Create a list of objects of type Responder which have in this constructor

* ID number, username, password, gender, paid, age and income

*/

Responder res1 = new Responder(113, "Luke", "test", 'M', true, 32, 120000);

Responder res2 = new Responder(114, "John", "test", 'M', true, 32, 120000);

Responder res3 = new Responder(112, "Mark", "test", 'M', true, 28, 160000);

Responder res4 = new Responder(115, "Mathew", "test", 'M', true, 40, 40000);

```
/**  
  
 * Add responders to a list of all Responder customers of system  
  
 */  
  
listResponders.add(res1);  
  
listResponders.add(res2);  
  
listResponders.add(res3);  
  
listResponders.add(res4);  
  
  
System.out.println("c) add a new customer to the dating service"+ "\n");  
  
  
/**  
  
 * c) add a new customer to the dating service  
  
 * Add new customer to the system  
  
 * Object type Responder, with ID, username, gender, isPaid, age, income  
  
 */  
  
Responder res5 = new Responder(115, "Charlie", "test", 'M', true, 25, 120000);  
  
System.out.println(res5.getUserName()+ " age:"+ res5.getAge()+ " was added to the  
system." + "\n");  
  
  
/**  
  
 * Add responders to a list  
  
 */
```

```
listResponders.add(res5);
```

```
/**
```

```
 * List all Responders
```

```
 */
```

```
System.out.println("e) display the details of all advertisers registered with the service" +  
"\n");
```

```
/**
```

```
 * e) display the details of all advertisers registered with the service
```

```
 */
```

```
currentUserAdv.printListAdvertisers(listAdvertisers);
```

```
System.out.println("\n" + "List of all responders:" + "\n");
```

```
System.out.println("f) display the details of all responders registered with the service" +  
"\n");
```

```
/**
```

```
 * f) display the details of all responders registered with the service
```

```
 */
```



```
currentUserRes.printListResponders(listResponders);
```

```
System.out.println("\n");
```

```
/**
```

```
 * Required. When new customers are created, i.e., when someone has paid and is
```

```
 * accepted, then they choose a new login and they are given their own default password
```

```
 * Get input username and password from user
```

```
 */
```

```
/**
```

```
 * input a user name
```

```
 */
```

```
String username = "Mark";
```

```
/**
```

```
 * input a password
```

```
 */
```

```
String password = "test";
```

```
System.out.println("\n");
```

```
System.out.println("System is login in..." + "\n");

/**
 * Validate login
 */
for (Iterator<Responder> it = listResponders.iterator(); it.hasNext(); )
{
    Responder res = it.next();
    if (res.getUserName().equals(username))
    {
        if(password == res.getPassword())
        {
            loggedIn = true;
            currentUserRes.setUserId(res.getUserId());
            currentUserRes.setUserName(res.getUserName());
            currentUserRes.setGender(res.getGender());
            currentUserRes.setAge(res.getAge());
            currentUserRes.setIncome(res.getIncome());
        } else
        {

```

```
        System.out.println("Wrong password");
    }
}

} // End login validation

System.out.println(currentUserRes.getUserName() + " is loggedIn");

System.out.println("\n");

// End login

/**
 * Stores Get responder age to check matches
 */
int responderAge = res2.getAge();

/**
 * Stores Get responder income to check matches
 */
double responderIncome = res2.getIncome();

/**
```

```
* Get responder gender to check matches
```

```
*/
```

```
char responderGender = res2.getGender();
```

```
System.out.println("b) get some matches for a responder...");
```

```
/**
```

```
* b) get some matches for a responder.
```

```
* loop through all advertisers
```

```
*/
```

```
for (int i=0; i<listAdvertisers.size();i++)
```

```
{
```

```
    // for each advertirser get their description of partner sought
```

```
    char desiredGender = listAdvertisers.get(i).getDesiredGender();
```

```
    int minDesiredAge = listAdvertisers.get(i).getDesiredMinAge();
```

```
    int maxDesiredAge = listAdvertisers.get(i).getDeridedMaxAge();
```

```
    double minDesiredIncome = listAdvertisers.get(i).getDesiredMinIncome();
```

```
    double maxDesiredIncome = listAdvertisers.get(i).getDesiredMaxIncome();
```

```
    // Check if the responder logged in match de gender
```

```
    if (responderGender == desiredGender)
```

```

{
    // check if gender responder match desired by advertiser

    if (responderAge >= minDesiredAge && responderAge <= maxDesiredAge)
    {
        // check if income responder match desired by advertiser

        if (responderIncome >= minDesiredIncome && responderIncome <=
maxDesiredIncome)
        {
            //add advertiser to a list o advertisers from a Responder

            listMatched.add(listAdvertisers.get(i));
        }
    }
}

if (listMatched.size() == 0) // check if zero matches were found
{
    System.out.println("Tough luck, no matches today");

} // End checking responder matches


// Responder interface after login


// Print greetings and username

```

```
System.out.println( "\n");

System.out.println("Logging in Responder....");


System.out.println("#####");


System.out.println( "Hello "+ res2.getUserName()+ "\n" );


/**
 * counts number of matches that a Responder have
 */

int totalMatches = 0;


/**
 * Get size of ArrayList that contains advertisers from a responder
 */

listMatched.size();


// Print total of matches

System.out.println("You have " + listMatched.size() + " matches" );


// Print list of all matches
```

```
System.out.println("\n" + "List of your matches" + "\n");
```

```
/**
```

```
 * Check if list is not empty before print all matches on the console
```

```
 */
```

```
if (listMatched.size() > 0) {
```

```
/**
```

```
 * Loops through the list and print matches on the console
```

```
 */
```

```
for (int i=0; i<listMatched.size();i++)
```

```
{
```

```
    System.out.println("##### Match "+ i + " #####");
```

```
    String value = listMatched.get(i).getUserName();
```

```
    System.out.println(i + " - " + value);
```

```
    int age = listMatched.get(i).getAge();
```

```
    System.out.println("Age: " + age);
```

```
    value = listMatched.get(i).getTextAdvert();
```

```
    System.out.println("Text advert: " + value + "\n");
```

```
// End listing responder matches

}

// Start process of sending message

/**
 * Get Responder name
 */
String resName = res1.getUserName();

System.out.println("b) ...choose one match and send the match a message...");

/**
 * Get responder input for choosing partner
 * The number is presented to the Responder in their user interface(console)
 * b) ...choose one match and send the match a message...
 */
int chosenPartner = 1;

/**
```



```

* This variables will hold 2 different text messages

*/

String loveMessage = "I love you so much";

String loveMessage2 = "Lets get married";


System.out.println( "\n");

System.out.println("Sending to messages from " + resName + " to " +
listMatched.get(1).getUserName());


System.out.println("#####" +
"\n");

// Send messages to partner chosen


/**
 * Required. *A reply message has an owner (which is a responder) and some text.
 * The Advertiser object listMatched will get a number chosen by the responder
 * which will be the owner and will add the message to an array of messages
 * of that chosen partner.
 */

listMatched.get(chosenPartner).addMessage(resName, loveMessage);

listMatched.get(chosenPartner).addMessage(resName, loveMessage2);

```

```

// Advertiser user interface after login

//b) then login that advertiser to get the message

System.out.println("b)...then login that advertiser to get the message.");

System.out.println("\n");

System.out.println("Logging in Avertiser..." + "\n");


// Print user name on the console

System.out.println("#####");

System.out.println("\n" + "Hi " + listMatched.get(1).getUserName() + "\n");


System.out.println("##### INBOX
#####");

System.out.println("\n");

System.out.println("You have " + listMatched.get(1).m_inbox.size() + " messages");

System.out.println("\n");


/**
 * List all messages in advertiser inbox
 */

listMatched.get(1).printInbox();

```

```
System.out.println("#####  
#####");
```

```
System.out.println("");
```

```
// Delete message process
```

```
/**
```

```
 * Input from user to choose message
```

```
*/
```

```
int chosenMessage = 1;
```

```
/**
```

```
 * Delete chosen message
```

```
*/
```

```
adv2.deleteMessage(chosenMessage);
```

```
System.out.println("Deleting message " + chosenMessage + "..." + "\n");
```

```
System.out.println("Done, deleted!" + "\n");
```

```
System.out.println("Printing new INBOX" + "...");
```

```

System.out.println("");

System.out.println("#####");

System.out.println("\n" + "Hi " + listMatched.get(1).getUserName() + "\n");


System.out.println("##### INBOX
#####");

System.out.println( "\n");

System.out.println("You have " + listMatched.get(1).m_inbox.size() + " messages");

System.out.println( "\n");


/**
 * List all messages in advertiser inbox
 */

listMatched.get(1).printInbox();


System.out.println("#####
#####" + "\n");


//d) delete an existing customer from the dating service

System.out.println("d) delete an existing customer from the dating service" + "\n");


//Remove customer process

```

```
/**
 * Input user id to delete
 */
int idToDelete = 113;

/**
 * Delete chosen ID
 */
for (Iterator<Responder> it = listResponders.iterator(); it.hasNext(); )
{
    Responder res = it.next();
    if (res.getUserId() == idToDelete)
    {
        it.remove();
        System.out.println("Deleting responder " + res.getUserName() + "...");
    }
}

} // End delete user

System.out.println("\n" + "List of all responders after delete one:" + "\n");
```

```
/**  
 * List all Responders  
 */  
for (int i=0; i<listResponders.size();i++)  
{  
    String value = listResponders.get(i).getUserName();  
    System.out.printf(value + "\n");  
}  
System.out.println("\n");  
} // end of main method  
} // end of class m8
```

