

BLOOD BANK MANAGEMENT



TEAM MEMBERS:

Muskan Matwani
AU1741027

Yesha Shastri
AU1741035

BLOOD BANK MANAGEMENT

Topic- Blood Bank Management: information about blood groups, blood donors, requests for bottles of blood, issue and receipt of blood bottles.

ABSTRACT:

Blood Bank Management aims at maintaining all the information pertaining to blood donors, different blood groups available in each blood bank, processing requests for bottles of blood, and help them manage in a better way. Creating such a system is essential for the modern times when everything needs to get done at a faster pace. It contributes to health care by helping the needy people avail blood at the fastest pace and hence, it saves more lives.

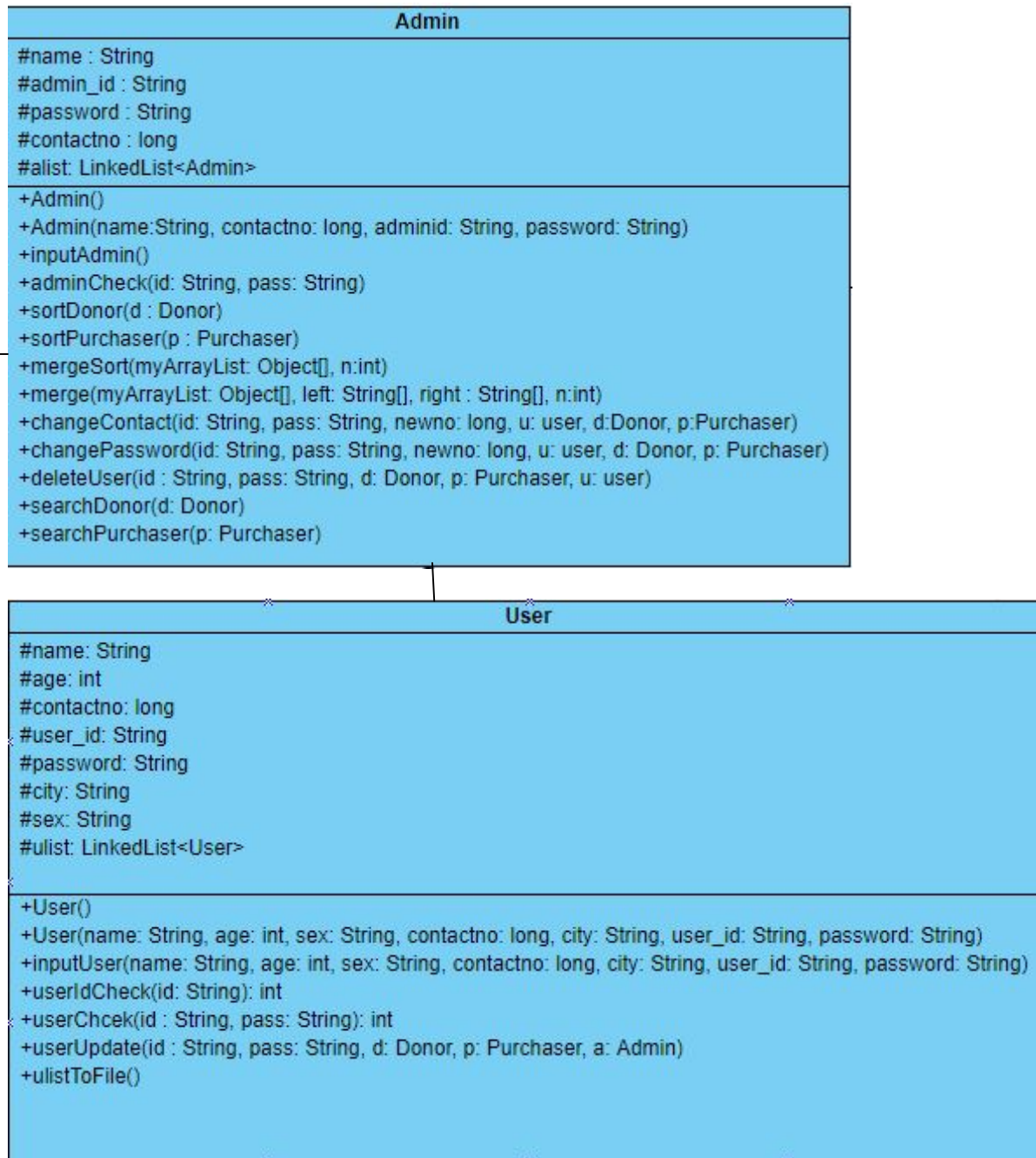
To build an efficient system, we will be using some basic data structures like linked list and array list which are dynamic in nature and hence would help us achieve our aim.

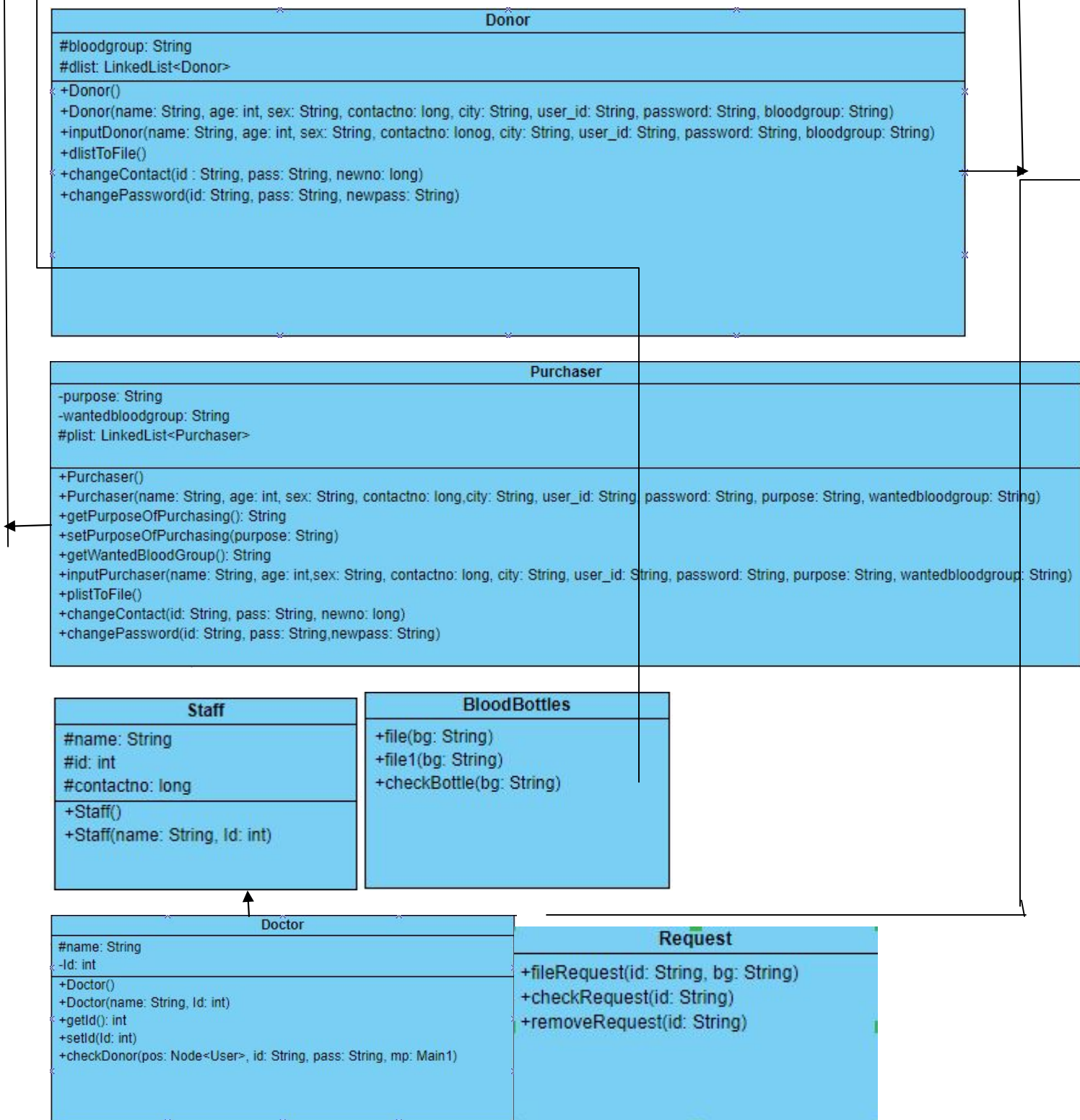
PROBLEM STATEMENT / BRIEF DESCRIPTION:

The problem aims at building a blood bank management system which can be used to maintain records of all the blood groups available, blood donor details, blood requester details, pending requests for blood bottles and finally the data of issuing and producing receipt for purchased blood bottles. The program used to create this system will require dynamic memory since the number of users – donors or purchasers are not fixed. The use of linked list will therefore be useful in implementing the above-mentioned feature. Also, linked list will have absolutely no memory wastage and operations like insertion and deletion will be easier to perform. Next, the records also have to be stored somewhere for general information and retrieval in future, so the concept of file handling is relevant here. Now, our approach towards building this system is as follows –

The system is operated upon by two kinds of people – User and Admin. As the program begins, a welcome frame appears asking the person operating to choose between a user and an admin. A user can either login or register depending on whether he/she is operating for the first time or not. In the case of admin, the respective person will only have the option of login and not register since the number of people who belong to admin panel will be pre-determined. Starting with the user, it is further divided into two distinctions –Donor and purchaser. The donor will have the functionality of donating blood and in order to do so he/she will have to respond to certain questions which are necessary for checking the basic blood health. Whereas, the purchaser can request for a specific blood group from the admin and if it is available then admin will give the receipt to the purchaser else admin will notify the purchaser whenever the blood bottle will be available. The functionality of inserting the details of users, deleting them when required, displaying all the details, searching for a specific donor or purchaser, sorting the details, and updating them are performed by the admin. All these functionalities have been implemented using linked list. It is done so by inserting the objects of the respective classes into linked lists having data type of those classes. To execute the functionalities, corresponding function to the functionality is called upon the object of the respective linked list. For the processing of data in all cases, firstly the existing data is read from the file and added to the linked list then after the data is added it is processed, and then it is finally written into the files. However, in some cases the data is only read depending on the functionality. The sorting functionality has been implemented using merge sort since it has the least, worst case running time $O(n \lg n)$. In sorting, arraylist is specifically used because it will be easy to first sort by one element of the linked list object and then order the complete linked list by putting the sorted object back into linked list. Therefore, when all the functionalities are combined, it will result into the full-fledged blood bank management system as stated by the problem. Further, to make it visually appealing and to provide ease for the user to operate upon, we have implemented the system on Graphic User Interface(GUI).

CLASSES AND RELATIONSHIPS:





LIST OF DATA- STRUCTURES USED

- **LINKED LIST** – It is a dynamic data structure used to store and process information. It has no memory loss other operations like insertion and deletion can be performed easily.
- **ARRAYLIST**- It is also a dynamic data structure which can process information efficiently. Though some memory loss is applicable, it is useful for the processing of information. This is used for sorting in our program.

ALGORITHMS:

- **INSERTION:** (inserting an object inside a Generic Linked List)

Void insert(T x) : inserting the object x with data type T

Step 1: START

Step 2: Creating an object q of class Node with the data x of type T

Node<T> q = new Node<T>(x)

Step 3: if first node is not null (first – data member of class Linked List), go to Step 5

Step 4: Assign first = q, go to Step 9

Step 5: assign object pos of class Node = first

Step 6: while pos.getNext() = null, go to Step 8

Step 7: assign pos to next node of pos, go to Step 6

Step 8: call setNext function to set q as the next node of pos

pos.setNext(q)

Step 9: STOP

1. Using insertion to insert objects of different classes (like User, Donor, and Purchaser)

Step 1: START

Step 2: Creating a linked list of User type

LinkedList<User> ulist = new LinkedList<>();

Step 3: Reading User details like name, gender, age, city, id, password

Step 4: Calling insert function and inserting the object of User type in the Linked List

ulist.insert(new User(name,.....,password))

Step 5: Writing ulist into the file 'UserDetails.txt'

Step 6: Appending 'Users.txt' with the id and password of the new User

Step 6.1: Make a FileWriter class object fw and PrintWriter object w (file – 'Users.txt')

Step 6.2: call write function of PrintWriter class and write the id and password of user into the file

Step 6.3: Close the file

Step 7: STOP

NOTE: Similarly for insertion Donor and Purchaser list

DELETION: Deleting an object from the generic linked list

Void remove (Node<T> pos)

Step 1: START

Step 2: Read an object pos of class Node having data type T

Step 3: If the object is not present at first node then go to Step 5

Step 4: Set the first node to the next node of pos

Step 5: Create a new object prev of class Node and set it equal to first node

Step 6: If the next node of prev is equal to pos then go to step 8

Step 7: Set prev equal to the next node of prev, go to Step 6

Step 8: Set the next node of prev as the next node of pos

Prev.setNext(pos.getNext())

Step 9: STOP

How it is implemented in our program?

It is used to delete the details of the user from User List

Step 1: START

Step 2: Read first node from the linked list of user and assign to pos

Step 3: If pos = null, go to Step 6

Step 4: If the id and password match then call the remove function for the object of userlinked list, go to Step 6

Step 5: Set pos to next node of pos, go to Step 3

Step 6: The updated data is written to the file

Step 7: STOP

NOTE: Implementing the above process to Donor and purchaser List as well.

Algorithm to delete that id and password from 'User.txt'

Step 1: START

Step 2: Declare String oldtext, aftertext, h and String array w

Step 3: Open the 'Users.txt' file in reading mode

Step 4: Read each line from the file and store it into string h. If end of file is reached, go to Step 8

Step 5: Assign w to contain all the strings present in the line h

W = h.split([\\s](#))

Step 6: If w[0] equals id and w[1] equals password of the User whose account is to be deleted, then go to Step 8 ,else go to Step 7

Step 7: Assign oldtext to oldtext plus h, and go to Step 4

oldtext = oldtext + h

Step 8: Read each line from the file and store it into string h. If the end of the file has reached, go to Step 10

Step 9: Assign aftertext to aftertext plus h

Aftertext = aftertext + h

Step 10: Close the file

Step 11: Open the file 'Users.txt' in writing mode

Step 12: Write oldtext and then write aftertext in the file

Step 13: Close the file

Step 14: STOP

UPDATION: Functionality which can update the details of user like contact and password

Step 1: START

Step 2: Read the first node from the linked list of user

Step 3: If the id and password match then set the contact number of the object equal to the newcontact number

Step 4: Set the object equal to next node of the object

Step 5: Repeat steps 2-4 till the end of linked list is reached

Step 6: The updated object is written back to file

Step 7: Print "CONTACT CHANGED SUCCESSFULLY!"

Step 8: STOP

Note: Same steps are used to update the password

- **SEARCHING** (Searching in the given list)
(Search by name)

Void searchDonor(Donor d, name n):

Step 1: START

Step 2: Set pos of type Node to the first node of dlist

Node<Donor> pos = d.dlist.getFirst()

Step 3: If pos is null, go to Step 8

Step 4: if data (Donor object) of pos does not have same name equal to n,
then go to Step7

Step 5: Set temp to 1

Step 6: Display Donor details of the data of node pos

pos.getData().toString()

Step 7: Set pos to next node of pos, go to Step 3

Step 8: If temp is not zero, go to Step 10

Step 9: Display “No Donor found”

Step 10: STOP

NOTE: Same search procedure is followed when search is made through age, blood group, etc.

- **SORTING** (Sorting the donor and purchaser list by increasing order of names)

Void sortDonor(Donor d)

Step 1: START

Step 2: Make an object al of class MyArrayList of String type

`MyArrayList<String> al = new ArrayList<>()`

Step 3: Set pos of Node type to the first node of dlist

Step 4: If pos = null, go to Step 7

Step 5: Add data's name of pos to the Array list al

`al.add(pos.getData().name)`

Step 6: Set pos to next node of pos, go to Step 4

Step 7: Call Merge Sort function to sort myArrayList

Step 8: Insert donor object in dlist in sequence with myArrayList

Step 9: Write dlist into the file 'UserDetails.txt'

void mergeSort(Object[] myArrayList , int n)

Step 0: START

Step 1: If $n < 2$, go to Step 12

Step 2: Define String array left and right of size $n/2$ and $n-n/2$ respectively

Step 3: Set l to 0

Step 4: If l = length of the left array, then go to Step 6

Step 5: Set left[i] to myArrayList[i]

Step 6: Set l to l+1, go to Step 4

Step 6: Set j to 0

Step 7: If j = length of the right array, then go to Step 9

Step 8: Set right[i] to myArrayList[i]

Step 9: Set j to j+1, go to Step 7

- Step 9:** Call recursive mergesort on left array
- Step 10:** Call recursive mergesort on right array
- Step 11:** Call merge function on left and right arrays
- Step 12:** STOP

void merge(Object[] myArrayList, String[] left, String[] right, int n)

Step 1:START

Step 2: Set integer a, b and i to 0

Step 3:If $i \geq n$, go to Step 10

Step 4:If $b \geq \text{length of the right array}$ OR $(a < \text{length of left array AND } \text{left}[a].\text{compareToIgnoreCase}(\text{right}[b]) < 0)$, then go to Step 5, else go to Step 7

Step 5: Set $\text{myArrayList}[i]$ to $\text{left}[a]$

Step 6: Set a to $a+1$, go to Step 9

Step 7: Set $\text{myArrayList}[i]$ to $\text{right}[b]$

Step 8: Set b to $b+1$, go to Step 9

Step 9: Set i to $i+1$, go to Step 3

Step 10: STOP

- **DISPLAY** (displaying the contents of a linked list)

String toString()

Step 1: START

Step 2: Set String str to ""

Step 3: Set Node pos to first node

Step 4: If pos is null, go to Step 7

Step 5: Set str to str plus data of pos

Str = str + pos.getData()

Step 6: Set pos to next node of pos

Step 7: return str

SNAPSHOTS (INPUT/OUTPUT):

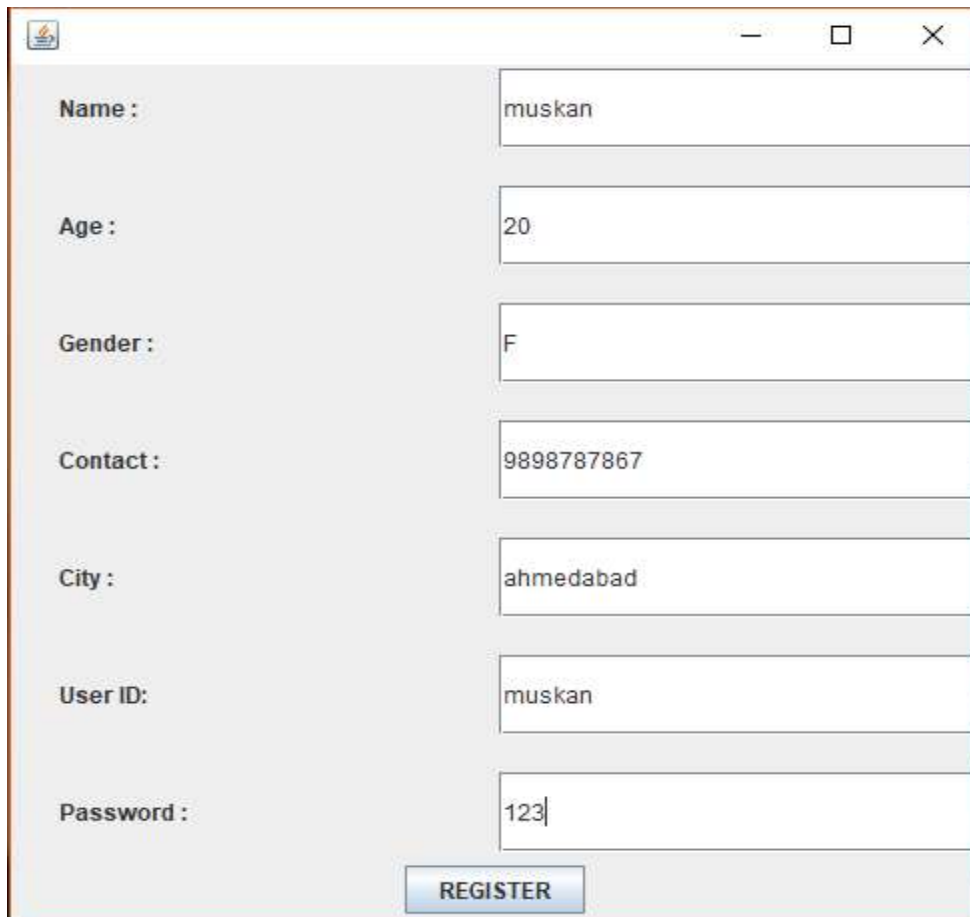
1. First window that appears asks whether the person is User or Admin



2. After clicking on User Button, the below frame asks user id and password if the person is already registered, or gives register option.



3. After clicking on REGISTER Button, below frame appears that asks the User details. After clicking on REGISTER button, The user is registered with the respective ID and password.



Name :	<input type="text" value="muskan"/>
Age :	<input type="text" value="20"/>
Gender :	<input type="text" value="F"/>
Contact :	<input type="text" value="9898787867"/>
City :	<input type="text" value="ahmedabad"/>
User ID:	<input type="text" value="muskan"/>
Password :	<input type="text" value="123"/>

4. Now, that user can login.



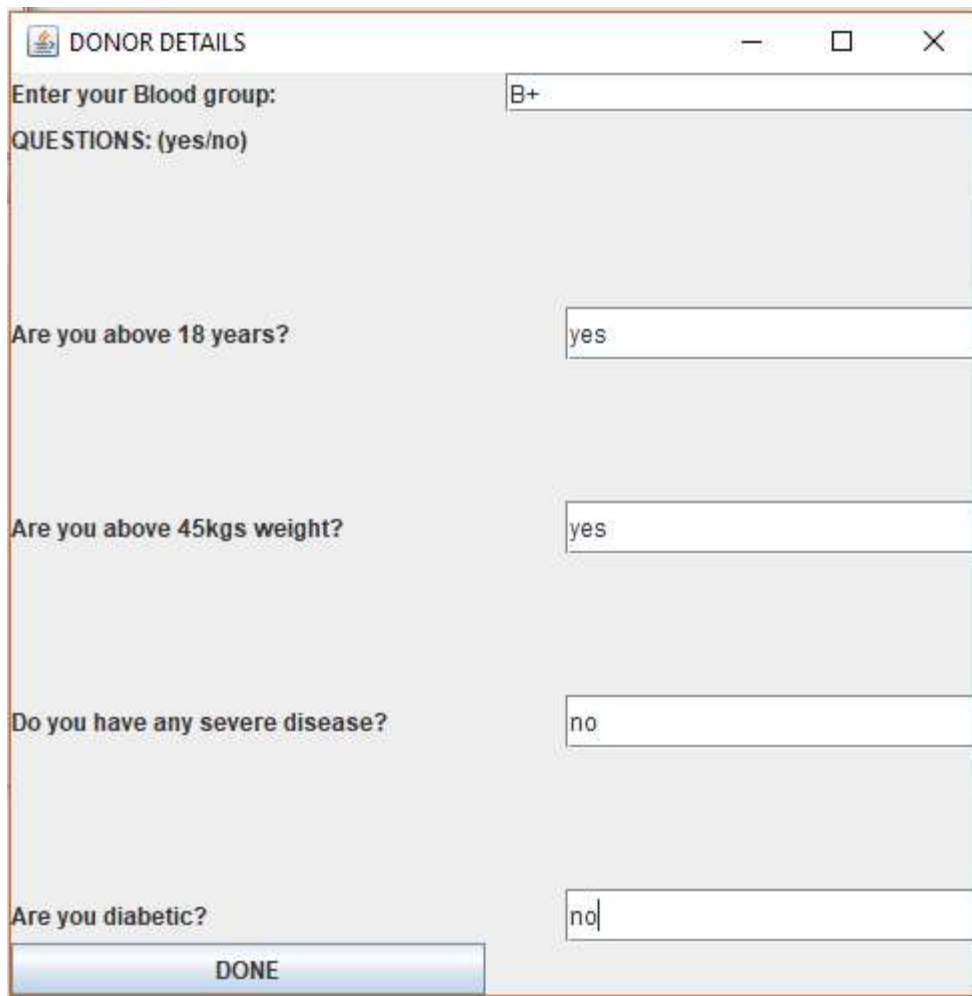
The screenshot shows a window titled "LOGIN WINDOW" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains two input fields: "User Id :" and "Password :". The "User Id" field contains the text "muskan". The "Password" field contains four hash symbols "####". Below the input fields are three buttons: "LOGIN", "REGISTER", and "BACK".

5. When the login is successful, it displays the below frame. It asks user to perform any of the following operations.



The screenshot shows a window titled "USER FUNCTIONS WINDOW" with a standard Windows-style title bar. The window has a header area with the text "Why Are You here?". Below this header, there are six buttons arranged in a 2x3 grid. The top row contains buttons labeled "DONATE", "REQUEST", and "UPDATE DET...". The bottom row contains buttons labeled "DELETE ACC...", "LOGOUT", and "BACK".

6. When the User clicks on “DONATE”, the below frame appears that checks whether the DONOR is fit to donate or not. If yes, then blood is donated and donor is added in the donor list and ‘DONORS.TXT’ file



The screenshot shows a window titled "DONOR DETAILS" with a standard Windows-style title bar (minimize, maximize, close buttons). The window contains a form with the following elements:

- A label "Enter your Blood group:" followed by a text input field containing "B+".
- A label "QUESTIONS: (yes/no)" followed by a large, empty text area.
- A label "Are you above 18 years?" followed by a text input field containing "yes".
- A label "Are you above 45kgs weight?" followed by a text input field containing "yes".
- A label "Do you have any severe disease?" followed by a text input field containing "no".
- A label "Are you diabetic?" followed by a text input field containing "no".
- A blue button labeled "DONE" at the bottom left of the form area.

7. When we click on “REQUEST” of frame 5, below window appears that asks for the details.

REQUESTER DETAILS

Purpose of Purchasing: need

DATE (till you want the blood group) : (dd/mm/yyyy) 10/11/2018

Wanted Blood Group : (A+ / A- / B+ / B- / AB+ / AB- / O+ / O-) A+

DONE

8. If the blood bottle is available, It displays “RECEIPT” , else prints “SUFFICIENT BOTTLES NOT AVAILABLE”

RECEIPT

=====
Receipt
=====

Blood group you wanted : A+

Price: Rs 1000

9. When we click on “UPDATE” on frame 5, below window appears that asks the user – what needs to be updated :



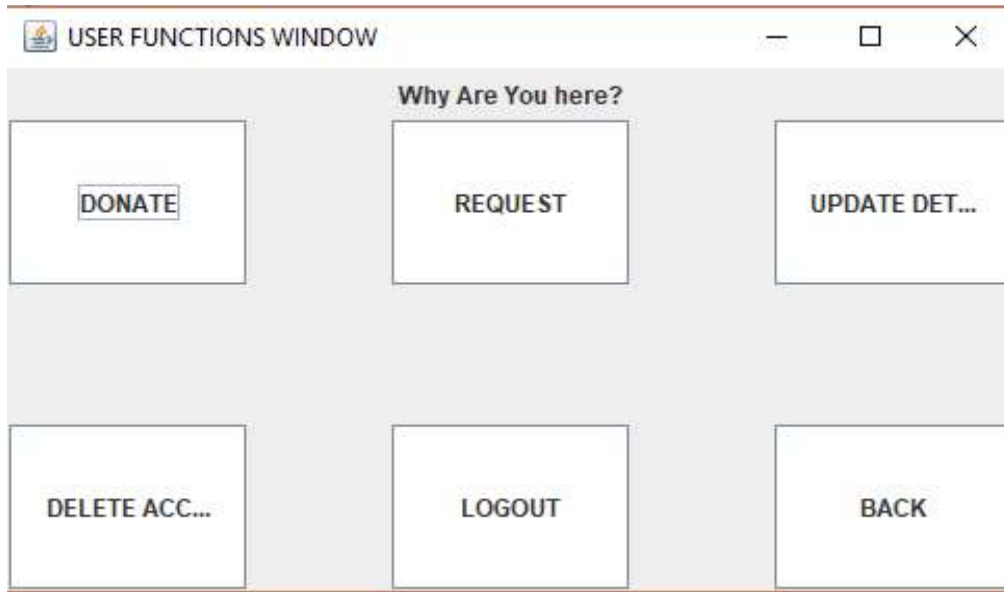
10. If the User clicks on “CONTACT”, in the above frame, It opens the below window:



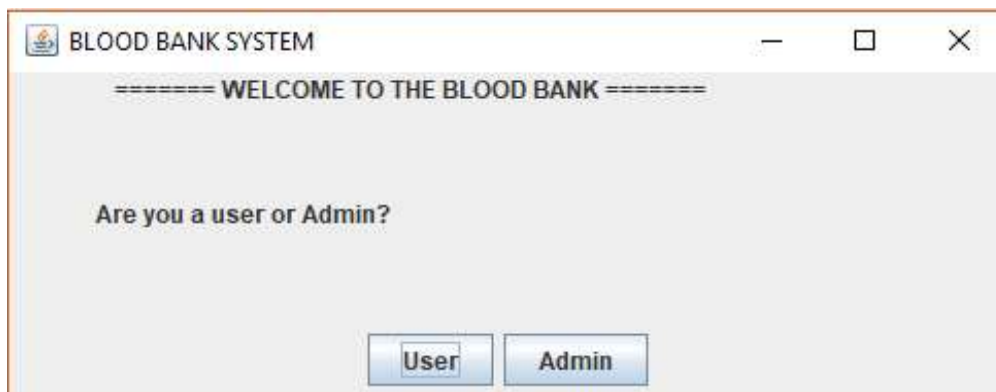
After clicking on “DONE”, the contact is updated.

Similarly, Password is also updated (whenever required).

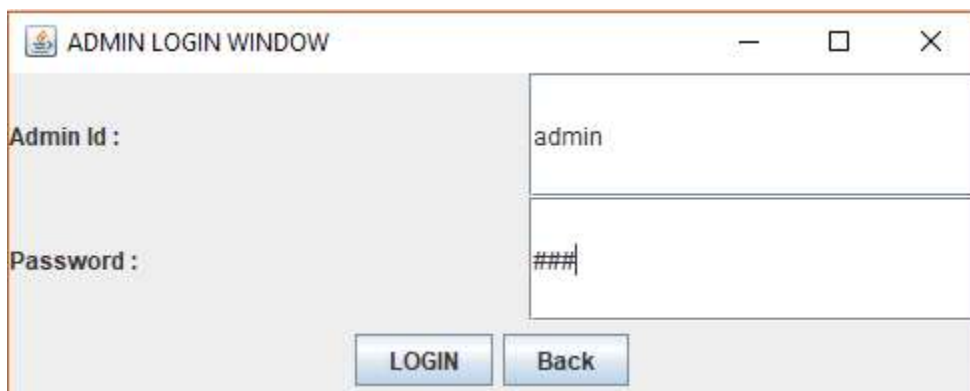
11. On clicking on “DELETE”, account of tat User is deleted from the system.



12. If we click on “ADMIN” , in the below frame, 13 frame appears:

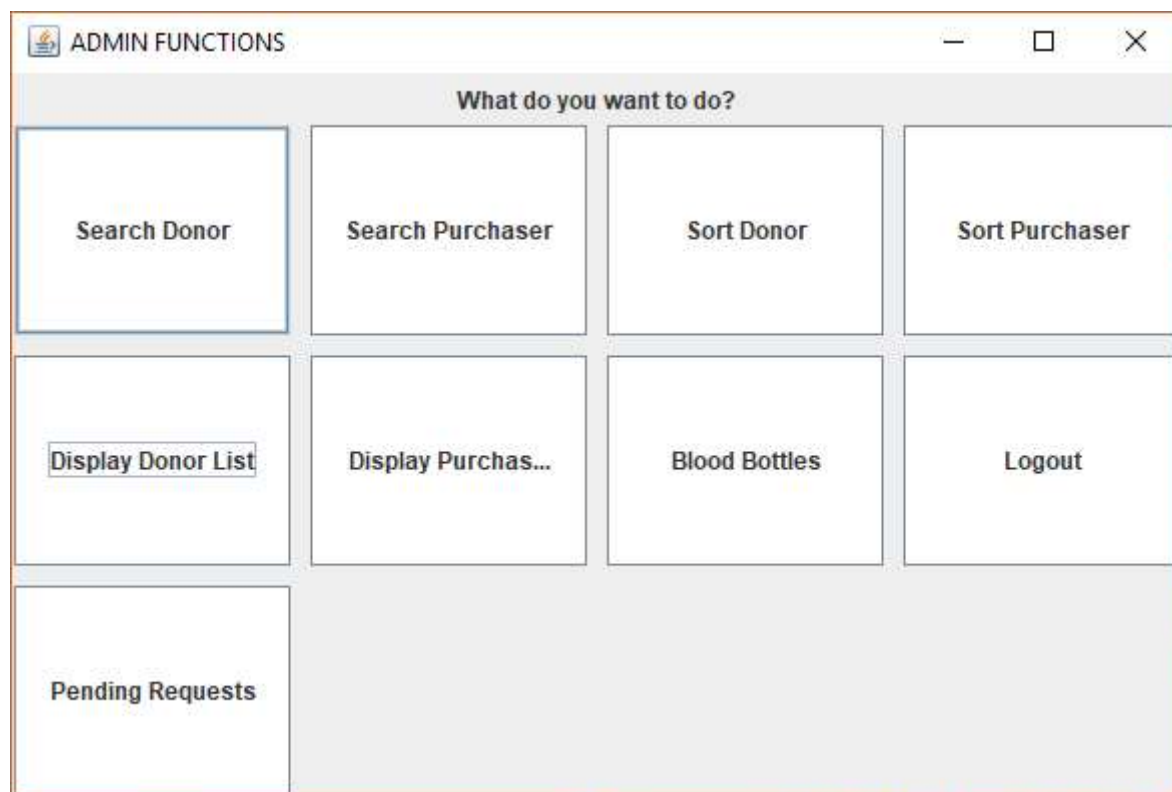


13. Below frame is “ADMIN LOGIN WINDOW”.



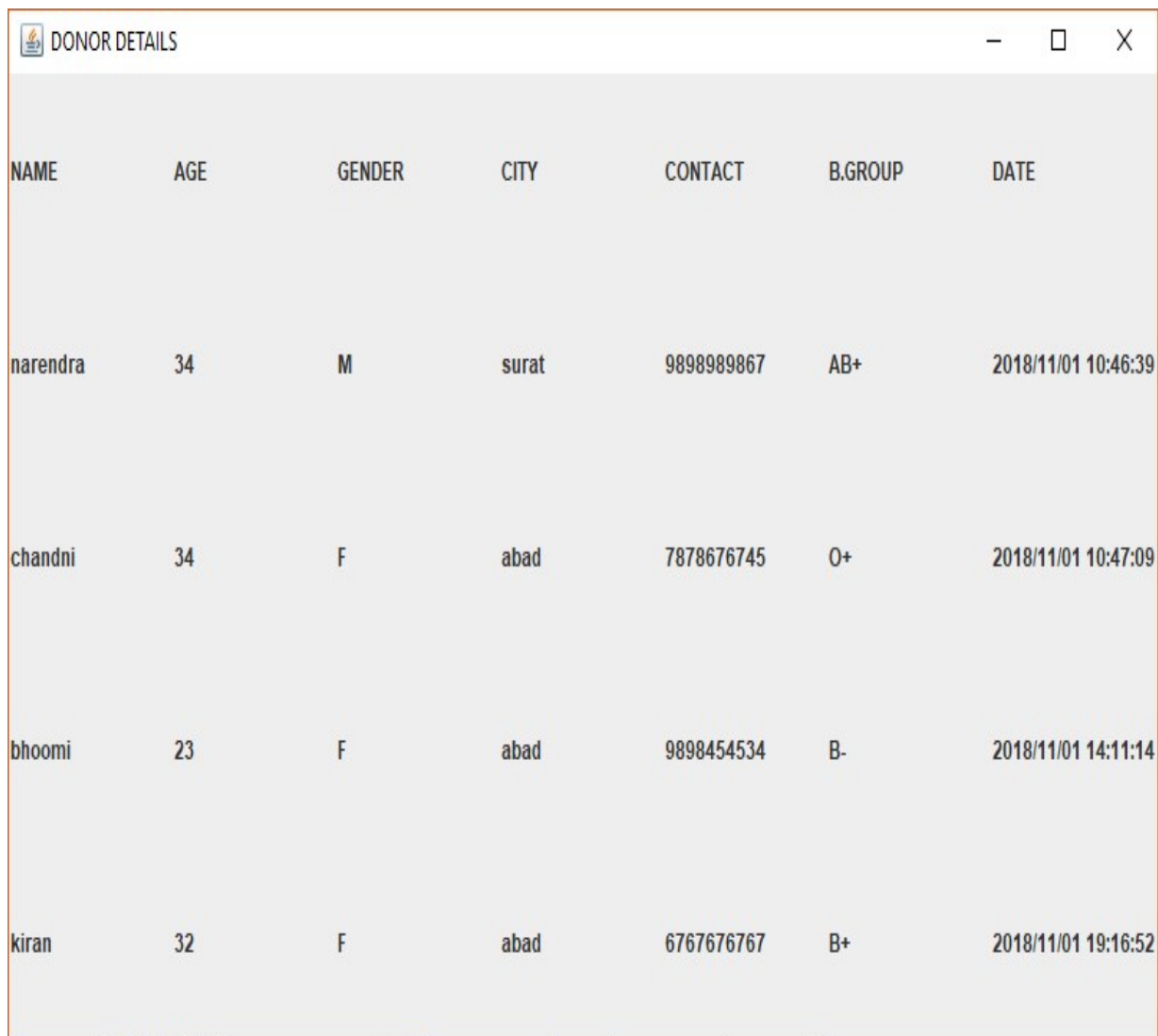
A screenshot of a software window titled "ADMIN LOGIN WINDOW". The window has a standard title bar with minimize, maximize, and close buttons. The main area is divided into two sections: "Admin Id :" and "Password :". The "Admin Id :" field contains the text "admin". The "Password :" field contains three hash symbols "###". Below these fields are two buttons: "LOGIN" and "Back".

14. If the login of admin is successful, below window appears , that asks the admin to perform any of the following functions.



A screenshot of a software window titled "ADMIN FUNCTIONS". The window has a standard title bar with minimize, maximize, and close buttons. The main area has a header "What do you want to do?". Below the header is a grid of buttons. The first row contains four buttons: "Search Donor", "Search Purchaser", "Sort Donor", and "Sort Purchaser". The second row contains four buttons: "Display Donor List", "Display Purchas...", "Blood Bottles", and "Logout". The third row contains one button "Pending Requests" on the left, and a large grey rectangular area on the right.


15.If the ADMIN clicks on “DISPLAY DONOR LIST”, below frame appears.



The screenshot shows a web application window titled "DONOR DETAILS". Inside the window is a table with the following columns: NAME, AGE, GENDER, CITY, CONTACT, B.GROUP, and DATE. The table contains four rows of donor data.

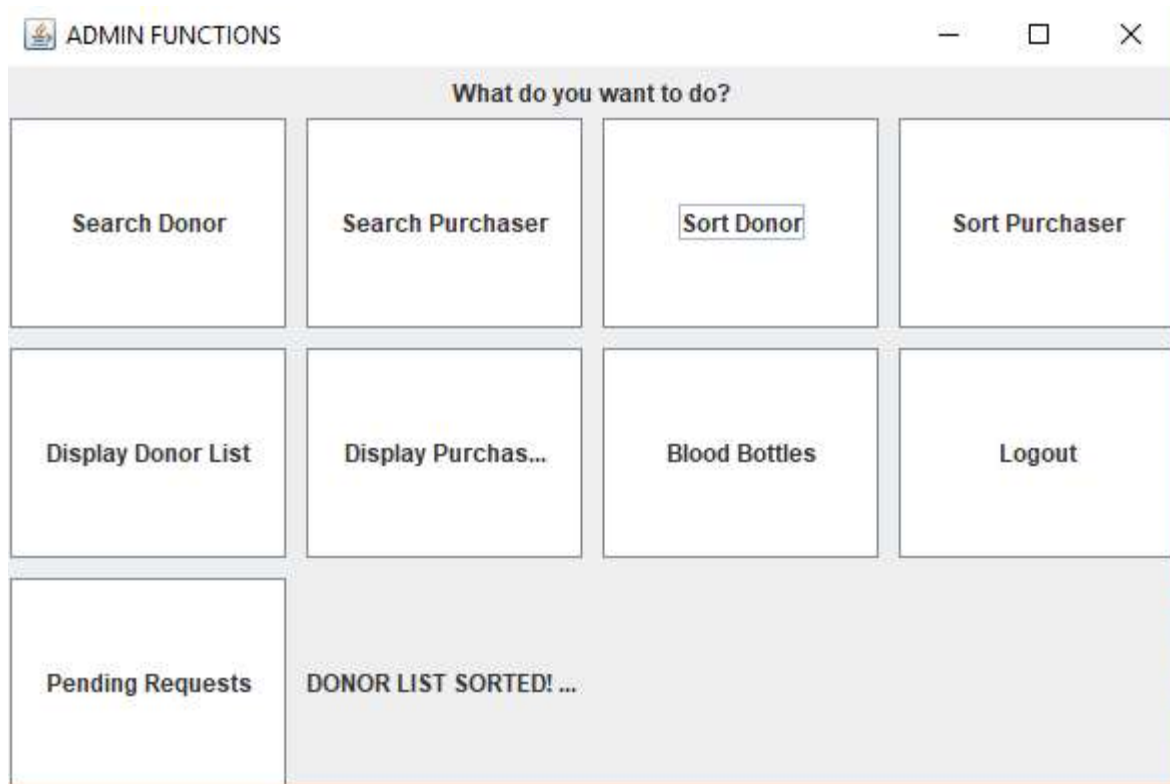
NAME	AGE	GENDER	CITY	CONTACT	B.GROUP	DATE
narendra	34	M	surat	9898989867	AB+	2018/11/01 10:46:39
chandni	34	F	abad	7878676745	O+	2018/11/01 10:47:09
bhoomi	23	F	abad	9898454534	B-	2018/11/01 14:11:14
kiran	32	F	abad	6767676767	B+	2018/11/01 19:16:52

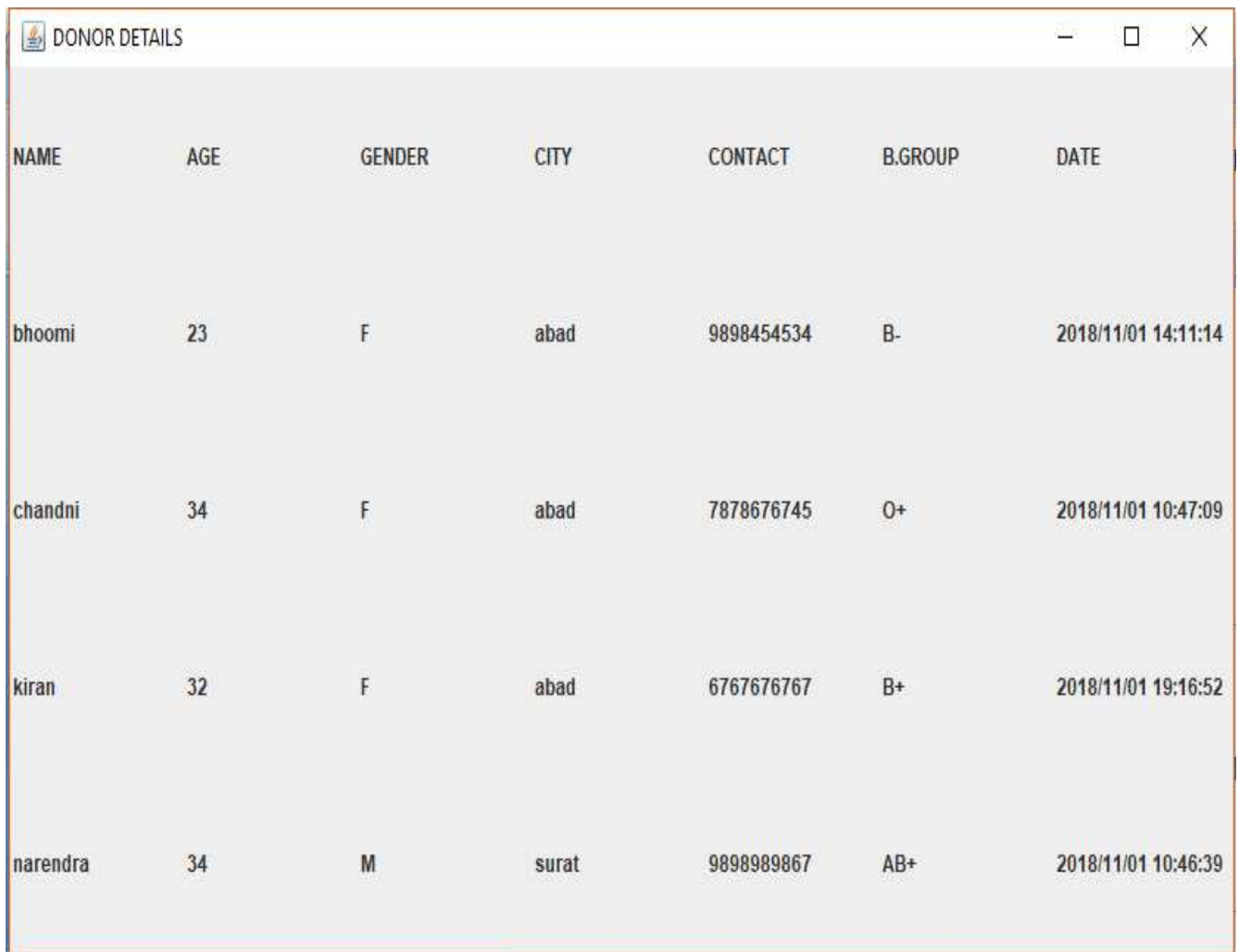
16. If the ADMIN clicks on “DISPLAY BLOOD BOTTLES” on frame 14 , below frame appears that displays the number of blood bottles available in the bank.



BLOOD GROUP	BOTTLES
A+	19
A-	2
B+	12
B-	1
O+	4
O-	2
AB+	0
AB-	3

17. If the ADMIN clicks on “SORT DONOR”, it displays that ‘donor list is sorted we can click on “DISPLAY DONOR LIST”” to verify.



18. DISPLAYING SORTED DONOR LIST.

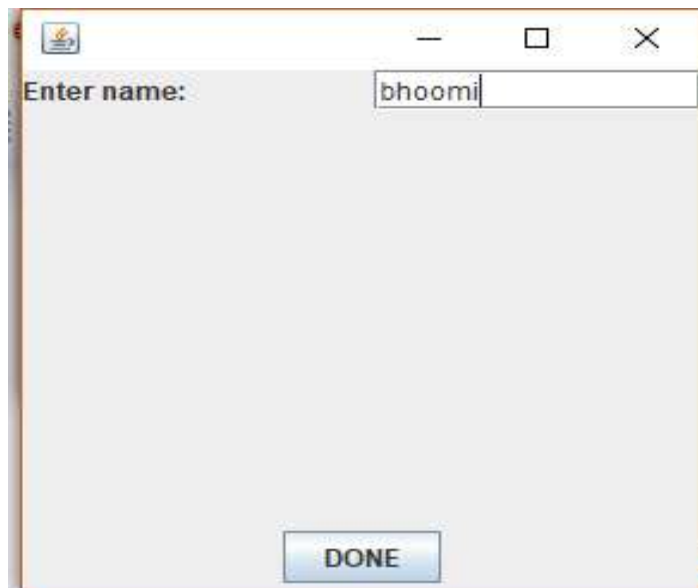
A screenshot of a software window titled "DONOR DETAILS". The window contains a table with seven columns: NAME, AGE, GENDER, CITY, CONTACT, B.GROUP, and DATE. There are four rows of donor data displayed. The window has standard OS controls (minimize, maximize, close) in the top right corner.

NAME	AGE	GENDER	CITY	CONTACT	B.GROUP	DATE
bhoomi	23	F	abad	9898454534	B-	2018/11/01 14:11:14
chandni	34	F	abad	7878676745	O+	2018/11/01 10:47:09
kiran	32	F	abad	6767676767	B+	2018/11/01 19:16:52
narendra	34	M	surat	9898989867	AB+	2018/11/01 10:46:39

19. If “SEARCH DONOR LIST” is clicked on frame 14, then it asks we need to search by which attributes as shown in the below picture.

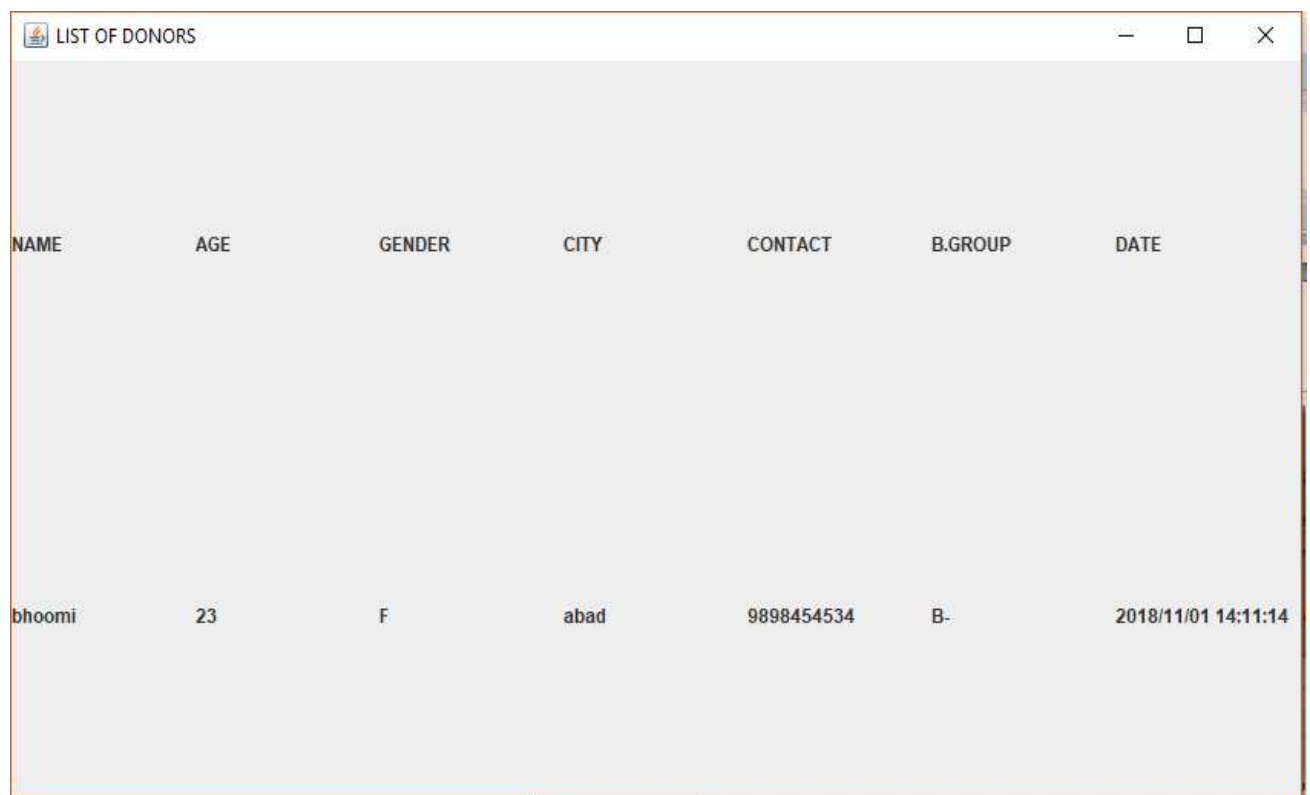


20. Since the search is by name “bhoomi”, new frame appears which shows whole details of that name.



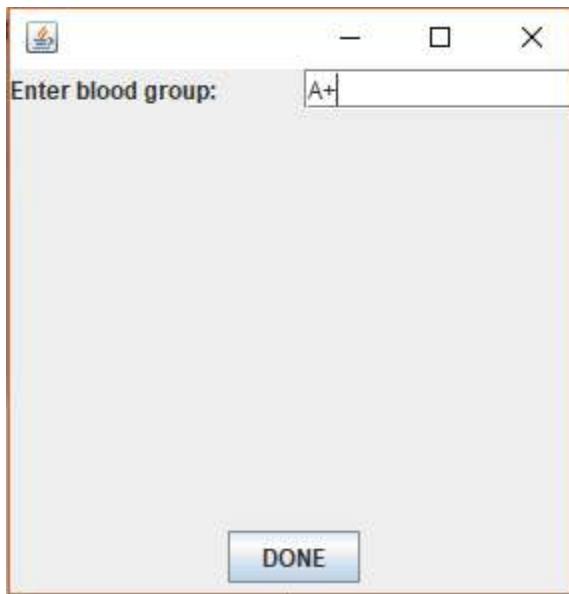
Enter name:

DONE



NAME	AGE	GENDER	CITY	CONTACT	B.GROUP	DATE
bhoomi	23	F	abad	9898454534	B-	2018/11/01 14:11:14

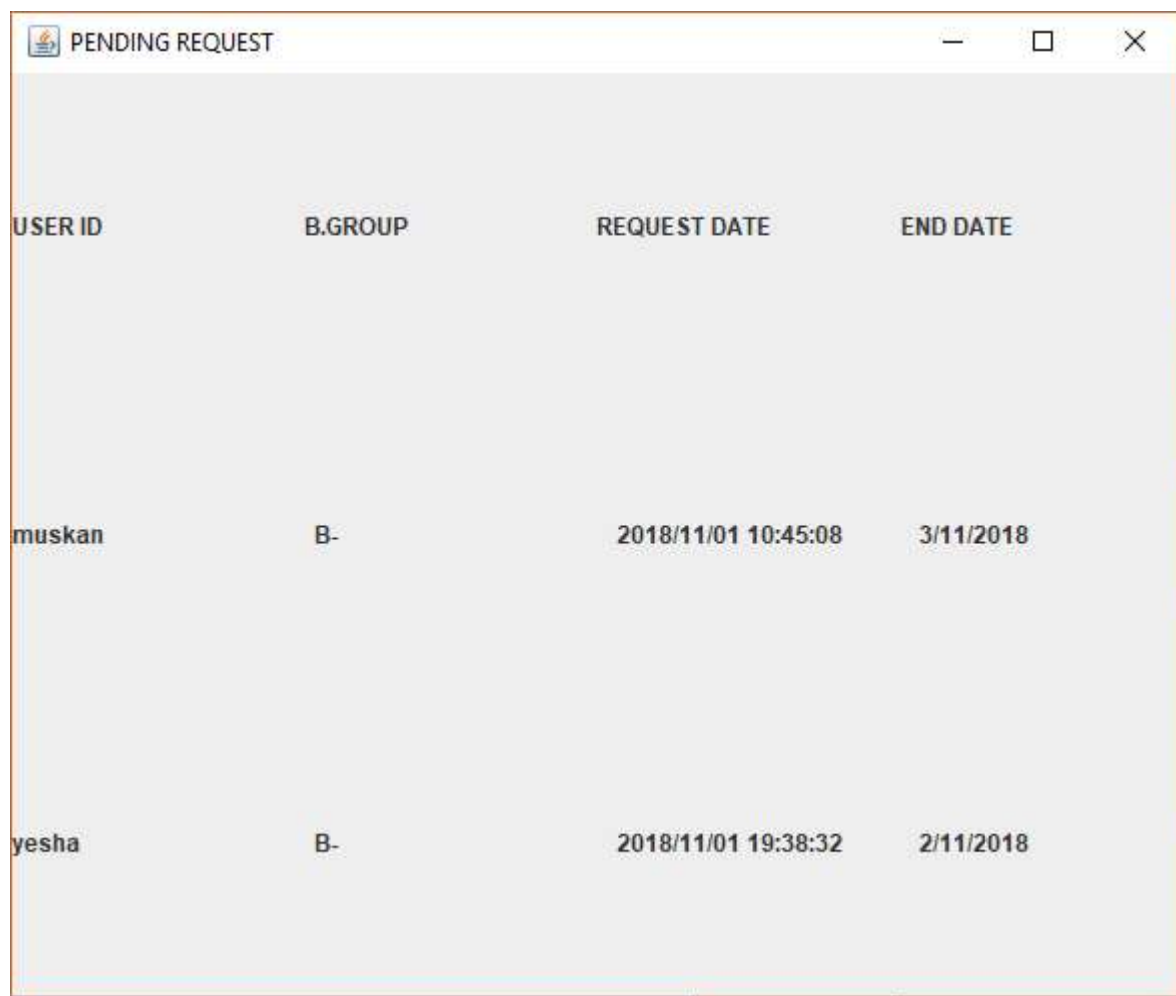
21. If we search by blood group say , “A+” , it shows this “NO BLOOD GROUP OF ‘A+’ FOUND.



No Donor with the BloodGroup 'A+' found

NOTE: Same functions are performed for SORT PURCHASER, SEARCH PURCHASER, and DISPLAY PURCHASER LIST

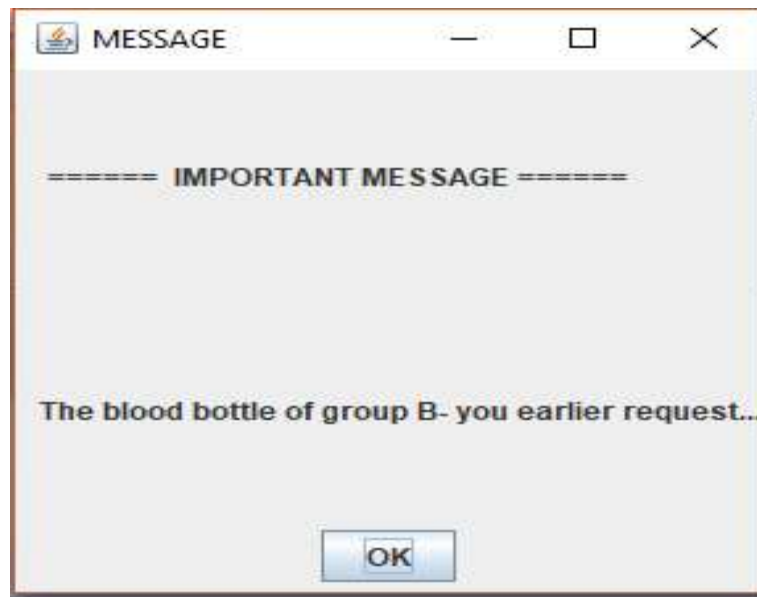
- 22.** When admin clicks on “PENDING REQUESTS” on frame 17, it displays user ids who were unable to get the blood groups, with their respective date of requesting and the last date till they want the blood group (the last date depicts the urgency the requester is having).



The screenshot shows a window titled "PENDING REQUEST" with a table containing two rows of data. The table has four columns: USER ID, B.GROUP, REQUEST DATE, and END DATE. The first row shows a request from 'muskan' for blood group 'B-' with a request date of '2018/11/01 10:45:08' and an end date of '3/11/2018'. The second row shows a request from 'yesha' for blood group 'B-' with a request date of '2018/11/01 19:38:32' and an end date of '2/11/2018'.

USER ID	B.GROUP	REQUEST DATE	END DATE
muskan	B-	2018/11/01 10:45:08	3/11/2018
yesha	B-	2018/11/01 19:38:32	2/11/2018

One more additional functionality is to display a pop up message when the blood group bottle that User requested (that was not available at that time), is now available, then the following pop up is appeared. The below message is only appeared to those ones who have urgency (depicted through their “end date”).



For instance: If there are 5 people requesting the same blood group when the bottles weren't available, then 3 donors donated that bottle, then only those 3 users who have more urgency (means their last date is closer than the other two), will be displayed the above message.

REFERENCES

- 1) "Applications of Linked List Data Structure." GeeksforGeeks, 30 Aug. 2018.
- 2) Pankaj, et al. "How to Implement ArrayList with Array in Java." JournalDev, 29 Mar. 2018.
- 3) "StreamCorruptedException: Invalid Type Code: AC." Stack Overflow.
- 4) Maneas, Sotirios-Efstathios. "Java.io.NotSerializableException – How to Solve Not Serializable Exception." Examples Java Code Geeks, 10 June 2014.