## 3. WRITTEN RESPONSES

### 3 a.

#### 3.a.i.

The overall purpose of the hospital reception program is to innovate the reception service of hospitals, which is usually done manually, by allowing it to be done by using a computing device.

#### 3.a.ii.

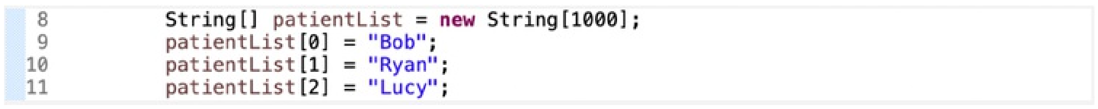
The functionality of the program is displaying a greeting phrase, “Welcome! ” + name to a patient that has visited the hospital before and has their name on the *patientList*, a list containing the names of patients who visited the hospital before, or displaying that it cannot find the patient’s information if the patient’s name is not stored on the *patientList*. Finally, the program adds the name of the patient that first visited the hospital to the *patientList*.

#### 3.a.iii.

The program’s inputs are the texts that answer whether the patient has visited the hospital before and the name of the patient. The output of the program is a text that displays greetings, the patient information could not be found, or the patient information is newly added to the program.

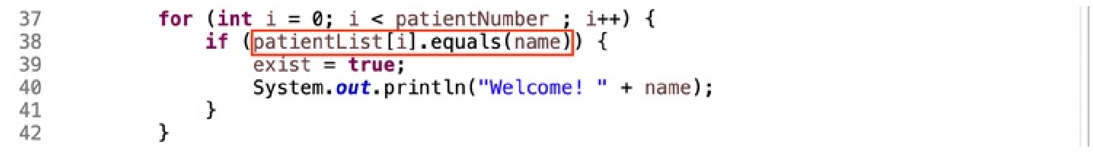
### 3 b.

#### 3.b.i.



```
        String[] patientList = new String[1000];
        patientList[0] = "Bob";
        patientList[1] = "Ryan";
        patientList[2] = "Lucy";
```

#### 3.b.ii.



```
        for (int i = 0; i < patientNumber ; i++) {
            if (patientList[i].equals(name)) {
                exist = true;
                System.out.println("Welcome! " + name);
            }
        }
```

#### 3.b.iii.

The name of the list is *patientList*.

#### 3.b.iv.

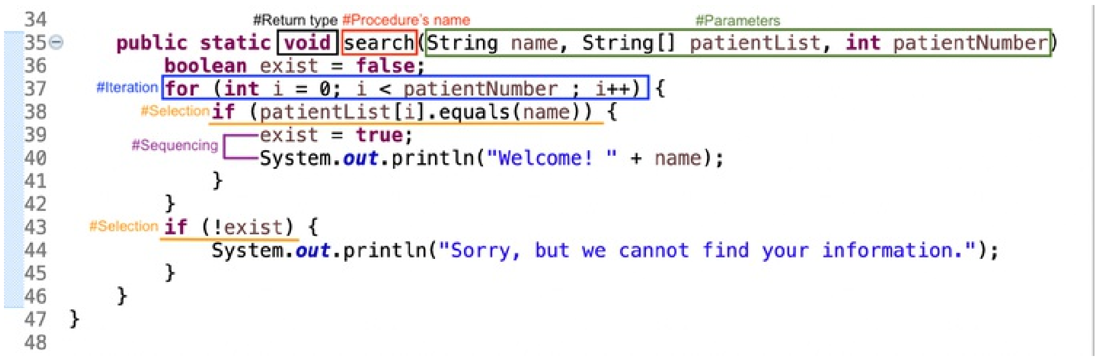
The data contained in the *patientList* represent the names of the patients that visited the hospital before. The *patientList* stores the names of the patients who visited the hospital before as string data.

#### 3.b.v.

The *patientList*
 manages complexity in the program code by allowing the names of patients to be stored as strings in one set of data. Without the
*patientList*
, the program would be written to store every single name of patients using different string variables. If the program works in that way, it would also be difficult to append the names of new patients to the program because new variables should be declared to store the names of new patients. Furthermore, not using a list when storing the names of patients would cause difficulty in the name searching functionality of the program because it is hard to search for the name that matches the text input using iteration when the names are in separate variables. Therefore, the
*patientList*
 manages the program's complexity by allowing data to be stored simply and making the searching functionality work correctly.

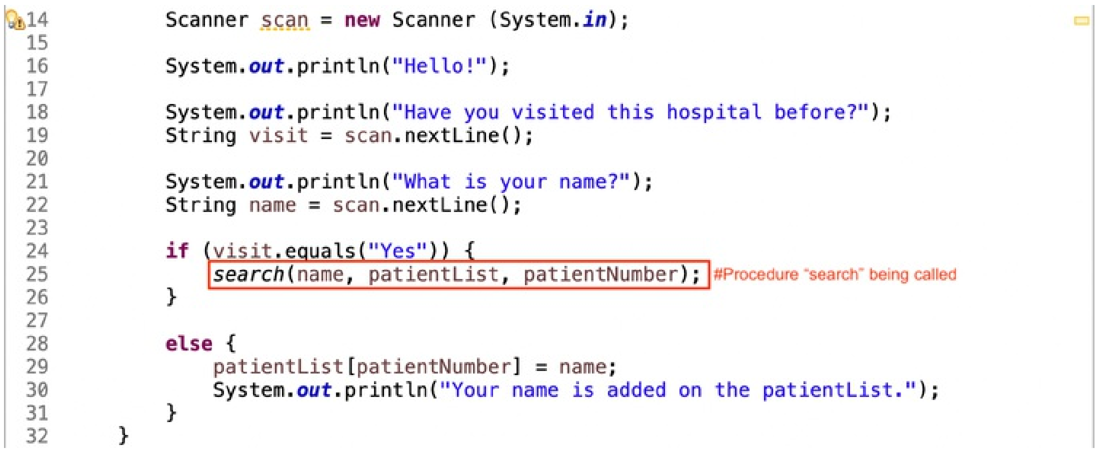
## 3 c.

### 3.c.i.



```java
    public static void search(String name, String[] patientList, int patientNumber) {
        boolean exist = false;
        for (int i = 0; i < patientNumber ; i++) {
            if (patientList[i].equals(name)) {
                exist = true;
                System.out.println("Welcome! " + name);
            }
        }
        if (!exist) {
            System.out.println("Sorry, but we cannot find your information.");
        }
    }
}
```

### 3.c.ii.



```java
        Scanner scan = new Scanner (System.in);

        System.out.println("Hello!");

        System.out.println("Have you visited this hospital before?");
        String visit = scan.nextLine();

        System.out.println("What is your name?");
        String name = scan.nextLine();

        if (visit.equals("Yes")) {
            search(name, patientList, patientNumber);
        }

        else {
            patientList[patientNumber] = name;
            System.out.println("Your name is added on the patientList.");
        }
    }
```

### 3.c.iii.

If the answer to the question asking whether they have previously visited the hospital is not "Yes", the name of the patient types after that is added to the list. When the patient enters "Yes" to the question asking whether they have previously visited the hospital and types their name, the procedure
*search* takes the string *name*, integer *patientNumber*, and *patientList* as parameters and compares the string variable *name* with each of the data stored in the *patientList*. It displays "Welcome! " + *name* if the *name* is found in the
*patientList*
, but displays, "Sorry, but we cannot find your information." if it is not found. The procedure contributes to the program's overall functionality by comparing the input data with every element in the list and displaying the result of whether the patient's name is included in the
*patientList* or not.

### 3.c.iv.

The procedure *search* takes the string input called *name*, which represents the name of the patient, the integer *patientNumber*, which represents the number of patients, and the *patientList*
containing the names of patients as its parameters. It initially sets the boolean named *exist*
to false. Then, using iteration (for statement), the program checks every index of the *patientList*
and finds out whether there is an element in the *patientList* that matches the parameter *name*. If there is, the procedure *search* sets the boolean named *exist* to true and displays "Welcome! " + *name*
. If there isn't a matching element, the boolean named *exist*
remains false and the procedure displays "Sorry, but we cannot find your information."

## 3 d.

### 3.d.i.

First call:

In the first call, the procedure *search* was tested with the case that the patient answered they had previously visited the hospital and input a name that already existed in the *patientList*. The procedure executes the first if statement that sets the boolean *exist* to true and displays “Welcome! ” + *name* under the condition when the string equal to the text name input is found on the *patientList*.

Second call:

In the second call, the procedure *search* was tested with the case that the patient answered they had previously visited the hospital but input a name that is not in the *patientList*. The procedure executes the second if statement that displays “Sorry, but we cannot find your information.” under the condition of when the boolean *exist* remains false.

### 3 d.ii.

Condition(s) tested by first call:

The condition tested by the first call of the procedure *search* is that there is a string in one of the indexes of the *patientList* that is equal to the text name input.

Condition(s) tested by second call:

The condition tested by the second call of the procedure *search* is that the boolean named *exist* remains false, which means that there is no name on the *patientList* that is equal to the text name input.

### 3.d.iii.

Results of the first call:

The result of the first call is that the boolean named *exist* changes to true, and the program displays “Welcome! ” + *name* as a text output.

Results of the second call:

The result of the second call is that the program displays the phrase "Sorry, but we cannot find your information." as a text output.