ICT2306- DATA STRUCTURES AND ALGORITHMS

Tithira Withanaarachchi- Batch 04

Lecturer:-
Mahesha Thejani

Contents

Task 1

Queue

Students' Waiting List -Queue

The above scenario is about a list of Students who are waiting to get an apartment from the university contact pool. According to the above scenario, the most suitable data structure for storing data about the students is the Queue.

Reason for Choosing the Queue :-

- ❖ Since each student should get the priority in order of the way they requested or the order they have been entered to the list.
- ❖ If Student has accepted an apartment, he/she can be removed from the front of the que using the deque() operation.
- ❖ If a student has rejected an apartment, they should be again added to the back of the data structure. Therefore, deque() can be used to remove the student from the front of the queue and again can be added to the back using enque() operation of the Queue.

The operations that can be done in a que are namely,

- enque()
- deque()
- Clear()
- peekFront()
- isFull()
- isEmpty()

These operations can be used in the scenario as below.

enque(element) -

By using the que , Students will always be added to the back of the list using the **enque()** method.

```
public boolean enque(Student st){//Method for entering to the back
  if(isFull()!=true){
   if(rear==maxSize-1)
   rear=-1;
   queArray[++rear]=st;//adding student to the back
   nStudents++;
   return true;
  }
  else{
      return false;
  }
}
```

```
Student sl=new Student(Integer.parseInt(idtxt.getText()), nametxt.getText(),sloctxt.getText(),AYtxt.getText(), Integer.parseInt(MAPtxt.getText()), Integer.parseInt(QTtxt.getText()));
boolean status=ql.enque(sl);
```

deque() -

Students who are in the list will be served and removed from the front of the que using the **deque()** method.

```
public Student deque(){//Method for dleleting from the front
    if(isEmpty()!=true){
    Student temp=queArray[front++];//removing student from the front
    if(front==maxSize)
        front=0;
    nStudents--;
    return temp;
    }
    else
        return null;
}
```

```
Student StuDeleted=ql.deque();
```

peekfront() -

When a student is being served, he/she is being searched through the **peekFront()** method and will check for a suitable apartment and assign them if he/she accepts it.

```
public Student peekFront(){// Method for searching the front element
    return queArray[front];
}
```

```
Student sl=ql.peekFront();
```

isFull() -

Before entering any student to the que, the availability for a location in the que will be checked through the **isFull()** method.

```
public boolean isFull(){//Method for checking weather the que is full
    return(nStudents==maxSize);
}
```

isEmpty() -

Before removing a student from the que, it will check whether any applicant is available in the que, using **isEmpty()** method.

```
public boolean isEmpty(){ //Method for checking weather the que isEmpty
    return(nStudents==0);
}
```

Task 2

Using Control Structures :-

```
//Button Click for inserting a new Student to the Que
private void InsertActionPerformed(java.awt.event.ActionEvent evt) {
    if("".equals(idtxt.getText())||"".equals(nametxt.getText())||"".equals(sloctxt.getText())||"".equals(AYtxt.getText())||"".equals(MAPtxt.getText())||"".equals(QTtxt.getText())){
        statustxt.setText("Any of the attributes cannot be empty");//Checking Whether the input fields are empty
    }
    else if(ql==null){//checking whether the que is Created
        statustxt.setText("Create the Que!");
    }
    else if(Integer.parseInt(QTtxt.getText())>100||Integer.parseInt(QTtxt.getText())<0){//Checking whether the Quality Score is within the range
        statustxt.setText("Quality Score shoulbe within 0 to 100");
    }
    else{
        boolean Eavailable=false;//checking whether the inserted ID is already available
        for(int i=0;i<ql.getnstud();i++){
            int id=ql.getque(i).getid();
            if(Integer.toString(id).equals(idtxt.getText())){
                Eavailable=true;
            }
        }

        if(Eavailable != true){
            Student sl=new Student(Integer.parseInt(idtxt.getText()), nametxt.getText(),sloctxt.getText(),AYtxt.getText(), Integer.parseInt(MAPtxt.getText()), Integer.parseInt(QTtxt.getText()));
            //Inserting the Student to the Que
            boolean status=ql.enque(sl);
            if(status==true){
            statustxt.setText(sl.getid()+" : "+sl.getnm()+ " has been Added to the Queue");
            }
            else{
            statustxt.setText("Insertion failed/Que is Full!");
            }
        }
        else{
            statustxt.setText("Enter a Different ID!");
        }
    }

}
```

1

2

3

1-

If we consider the main control structure which belongs to the square numbered as one, it validates whether the entered data to the text fields present in the form is within the range or gives a meaning to the related attribute. Other than that, it also checks whether the que has been created since it is an essential element in inserting a student object to a que.

2-

Square numbered as 2 represents a control structure which is used to check whether the number inserted as the ID of the student is already available. If that ID is already available, an error message will be displayed in the GUI. It Consist of a loop to go through all the object available to check whether a similar ID is there and an if statement to add the new Student object only if the entered ID is not already available. Otherwise, the application will display an error message.

3-

Square numbered as 3 represents the control structure used for choosing the correct message to display depending on the status of the insertion of a new student to the queue.

```
private void CheckAccomActionPerformed(java.awt.event.ActionEvent evt) {
  //Button Click for Checking for a matching apartment for student in the front of the Que
  statustxt.setText("");
  try{
   ArrayList<Apartment> ar=Apartmentfrm.alist;
   listl.removeAll();
    boolean ApartmentFound=false;
    for(int i=0;i<ar.size();i++){
        if((ar.get(i).getAvbty()==true) && (ar.get(i).getRent()<= ql.peekFront().getMAP()) && (ar.get(i).getQsc() >= ql.peekFront().getQT())){

            listl.add("ID :- " +ar.get(i).getid());
            listl.add("Location :- " +ar.get(i).getloc());
            listl.add("Maximum number of Rooms :- " +Integer.toString(ar.get(i).getMaxR()));
            listl.add("Availability:- " +Boolean.toString(ar.get(i).getAvbty()));
            listl.add("Rent :- " +Integer.toString(ar.get(i).getRent()));
            listl.add("Quality Score :- " +Integer.toString(ar.get(i).getQsc()));
            listl.add("----------------------------------------");
            ApartmentFound=true;
        }



    }
    if(ApartmentFound==false) {
       statustxt.setText("Apartment Not Found");
    }
  }catch(Exception ex){
      statustxt.setText("Check Apartment Failed");
  }

}
```

Here, the control Structures are used to filter the apartments according to the student's requirement. A loop is used to go through all the apartments and an "if" statement is used for checking whether each apartment satisfies the requirement of the student. If any of the apartment satisfy the requirement, it will be displayed in the user interface using a list.

```
private void alidActionPerformed(java.awt.event.ActionEvent evt) {
   //Button Click for allocating an apartment by ID for the Student in the front
    statustxt.setText("");
    ArrayList<Apartment> ar=Apartmentfrm.alist;
    boolean allocated=false;
    for(int i=0;i<ar.size();i++){
          if(ar.get(i).getid().equals(alID.getText())){

              ar.get(i).setAvbty(false);
              Student StuDeleted=ql.deque();
              if(StuDeleted != null){

              statustxt.setText("Apartment "+ar.get(i).getid()+ "has been allocated to "+StuDeleted.getid());
              }
              allocated=true;
              break;

          }



    }
     if(allocated==false){
        statustxt.setText("Apartment not Found");
     }
```

Here, Control statement is are used in order to assign an apartment for a student. This program will go through all the apartments using a loop and find the assign apartment to the student. Then it will make availability of that apartment equals false using an if conditional statement. Also, another conditional statement is used for displaying an error message if an apartment is not found with the entered ID by the user.

Examples for Exception Handling in the Code:-

```
private void CreateQActionPerformed(java.awt.event.ActionEvent evt) {
  //Button Click for Creating the Que
  statustxt.setText("");
  statustxtl.setText("");
  try{
  Queue qq=new Queue(Integer.parseInt(Qsize.getText()));
  ql=qq;
  statustxtl.setText("Que has been Created");
  }catch(Exception ex){
      statustxtl.setText("Error in creating the que: "+ex);
  }

}
```

Shown above is the code for creating the que. If an error occurred while creating the que in the try block, it will be caught by the catch block and display an error message.

```
private void SearchActionPerformed(java.awt.event.ActionEvent evt) {
   //button click for Searching the Student in front of the Que
    statustxt.setText("");
    try{
    Student sl=ql.peekFront();
    statustxt.setText(String.valueOf(sl.getid())+":- "+sl.getnm()+" is in the front of the Queue");
   }catch(Exception ex){
       statustxt.setText("Error in searching the Student: "+ex);
   }


}
```

Above code implemented in the Search button of the GUI will display the student in the front of the que using the peekfront() method in the Que class. If any error occurred while searching the student, it will be caught by the catch block and display an error message.

```
private void KeepWaitingActionPerformed(java.awt.event.ActionEvent evt) {
  //Button CLick for Keeping the Student in the front waited
    statustxt.setText("");
    try{
    Student st=ql.deque();
    ql.enque(st);
    statustxt.setText(st.getid()+":"+st.getnm()+ " has been added to the Back");
    }catch(Exception ex){
        statustxt.setText("Keep Waiting Failed");
    }
}
```

This is the code implemented in the button for “Keep Waiting” function, which will be used to remove the student from the front of the que using deque() method and put him/her back in the que using the enque() method. This option is used by the operator if the student rejects an Apartment suggested for him/her.

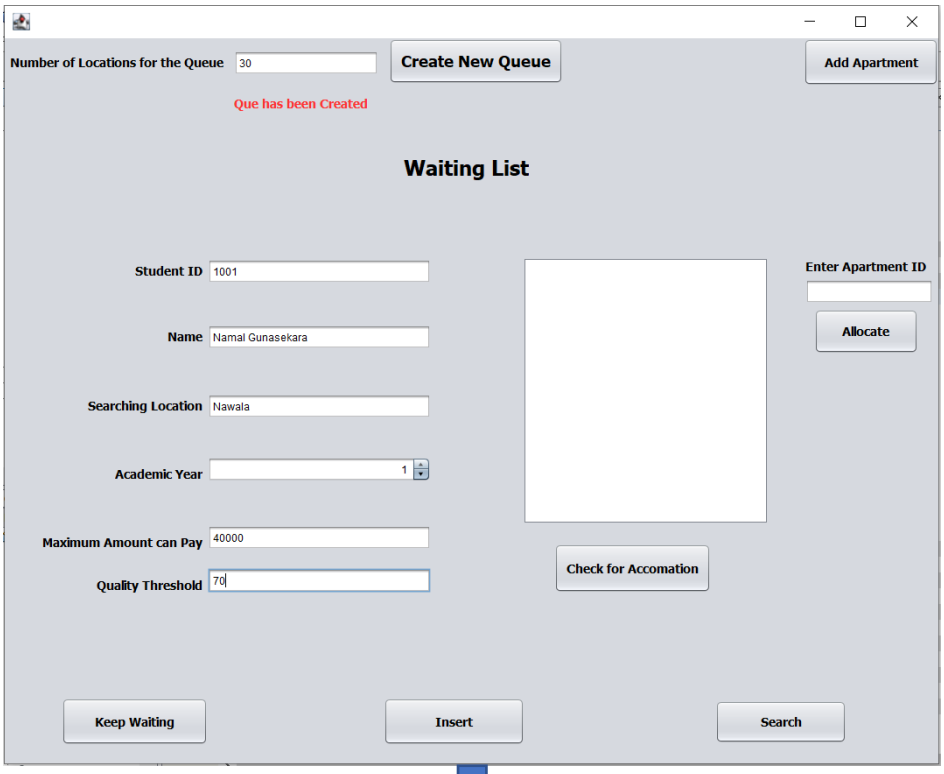
```
private void CheckAccomActionPerformed(java.awt.event.ActionEvent evt) {
  //Button Click for Checking for a matching apartment for student in the front of the Que
  statustxt.setText("");
  try{
   ArrayList<Apartment> ar=Apartmentfrm.alist;
   listl.removeAll();
    boolean ApartmentFound=false;
    for(int i=0;i<ar.size();i++){
        if((ar.get(i).getAvbty()==true) && (ar.get(i).getRent()<= ql.peekFront().getMAP()) && (ar.get(i).getQsc() >= ql.peekFront().getQT())){

            listl.add("ID :- " +ar.get(i).getid());
            listl.add("Location :- " +ar.get(i).getloc());
            listl.add("Maximum number of Rooms :- " +Integer.toString(ar.get(i).getMaxR()));
            listl.add("Availability:- " +Boolean.toString(ar.get(i).getAvbty()));
            listl.add("Rent :- " +Integer.toString(ar.get(i).getRent()));
            listl.add("Quality Score :- " +Integer.toString(ar.get(i).getQsc()));
            listl.add("----------------------------------------");
            ApartmentFound=true;
        }



    }
    if(ApartmentFound==false) {
       statustxt.setText("Apartment Not Found");
    }
  }catch(Exception ex){
      statustxt.setText("Check Apartment Failed");
  }

}
```

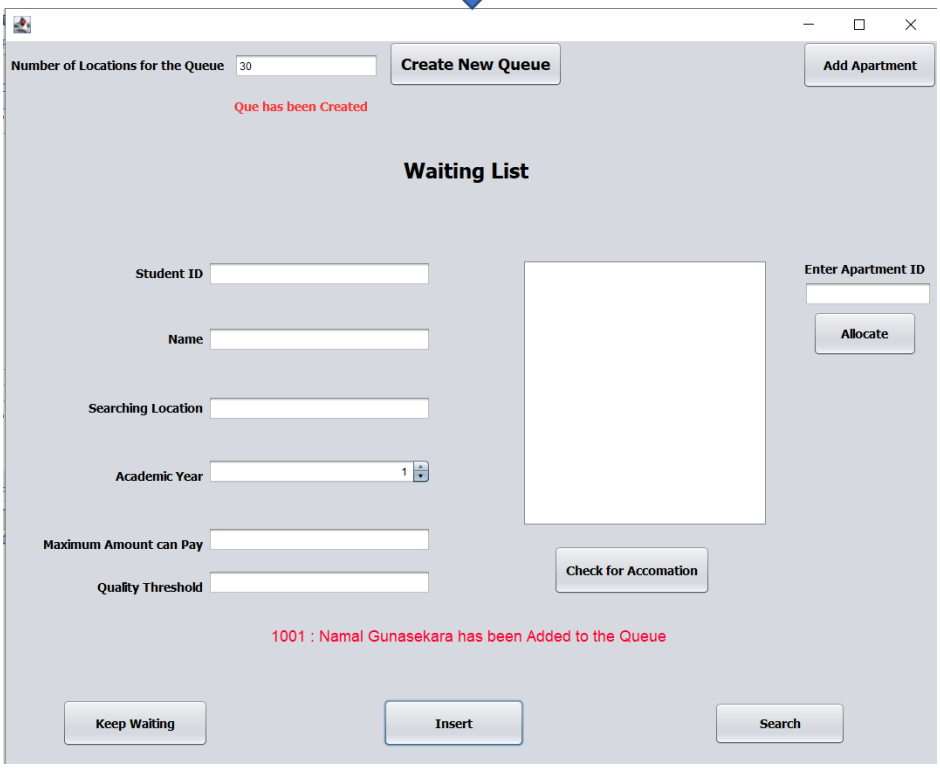
On the Above code, try-catch blocks are used to overcome the errors that may occur due to reasons other than not finding an Apartment which matches the Student.

Test Cases for the System Implemented.

Student Housing System	
Test Cases	
Test Unit: Insert Student	Tester: Tithira Withanaarachchi
Test Case ID: 01	Test Type: Black Box Testing
Description: Insert details about a Student who is expecting to get an Apartment to the que as a Student object.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date: 17/06/2021
Title: Inserting a new Student to the que.	Test Execution Time:

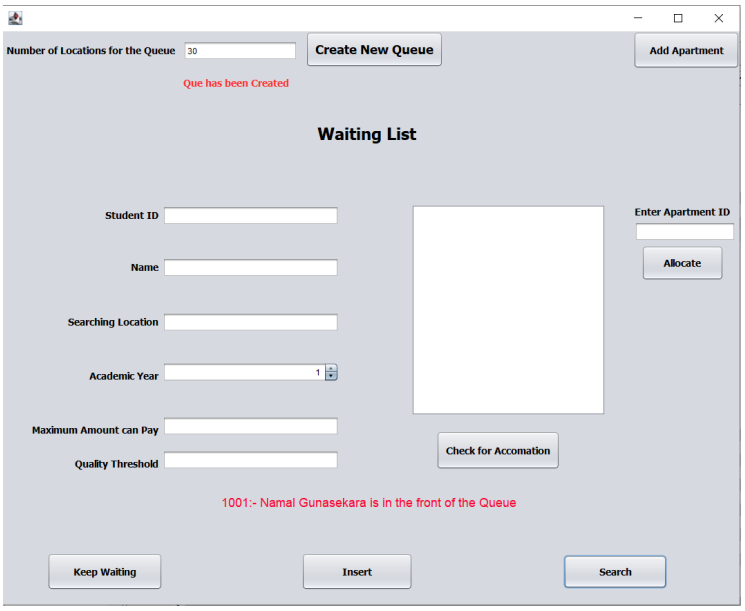
Step No	Test Steps	Test Data	Expected Results	Actual Result	Status(Pass/Fail)	Notes
01	Create a Que	Number of Locations= 30				
02	Insert Valid Data	Student ID= 1001, Name= Namal Gunasekara, Searching Location= Nawala, Academic Year = 1, Maximum Amount can pay= 40000, Quality Threshold= 75	Display "1001:- Namal Gunasekara has been added to the Que"	Display "1001:- Namal Gunasekara has been added to the Que"	Pass	


Number of Locations for the Queue
30
Create New Queue
Add Apartment
Que has been Created
Waiting List
Student ID
1001
Name
Namal Gunasekara
Searching Location
Nawala
Academic Year
1
Maximum Amount can Pay
40000
Quality Threshold
70
Enter Apartment ID
Allocate
Check for Accomation
Keep Waiting
Insert
Search


Number of Locations for the Queue
30
Create New Queue
Add Apartment
Que has been Created
Waiting List
Student ID
Name
Searching Location
Academic Year
1
Maximum Amount can Pay
Quality Threshold
Enter Apartment ID
Allocate
Check for Accomation
1001 : Namal Gunasekara has been Added to the Queue
Keep Waiting
Insert
Search

Student Housing System	
Test Cases	
Test Unit: Search Student.	Tester: Tithira Withanaarachchi
Test Case ID: 02	Test Type: Black Box Testing
Description: Search the student in front of the Que, who should be served next.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date: : 17/06/2021
Title: Searching the Student in front of the Que.	Test Execution Time:

Step No	Test Steps	Test Data	Expected Results	Actual Result	Status(Pass/Fail)	Notes
01	Create Que					
02	Insert Students to the Que					
03	Click on the Search Button		Display a message which has the structure "<id of the Student>:-<Name> is in Front of the Que"	Display a message which has the structure "<id of the Student>:-<Name> is in Front of the Que"	Pass	

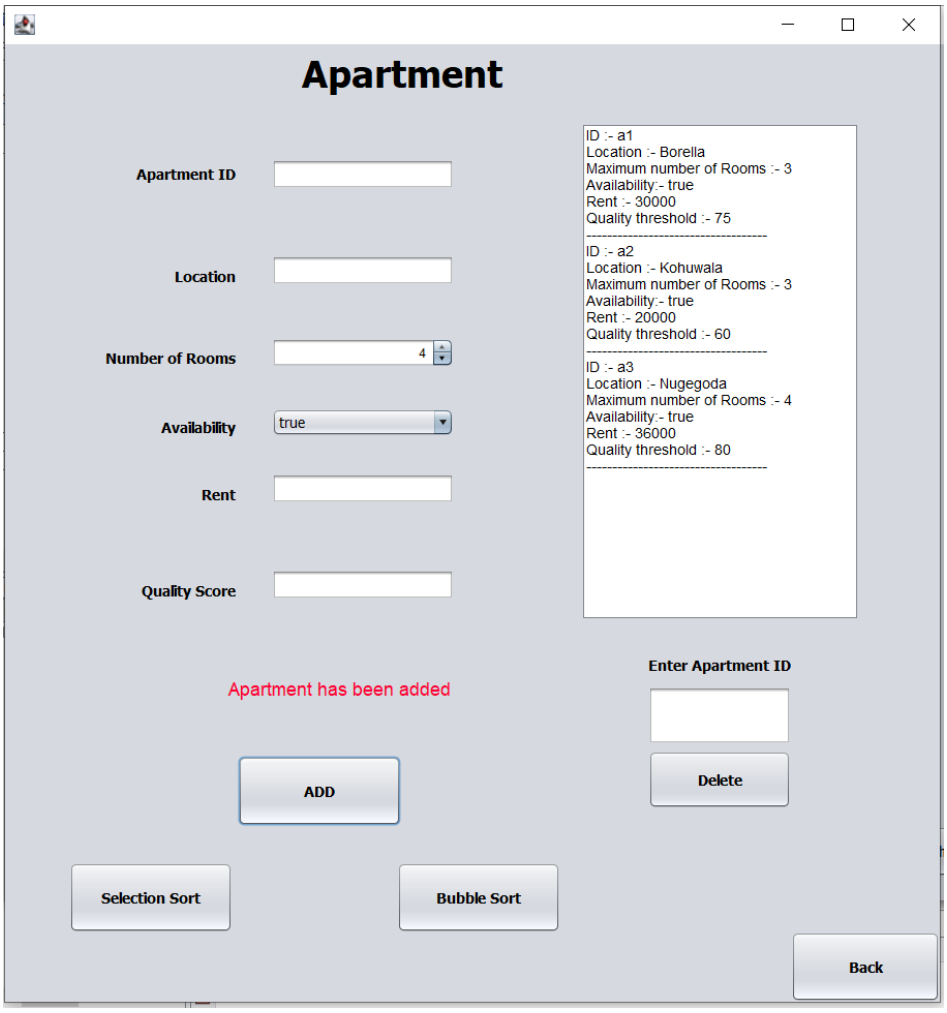


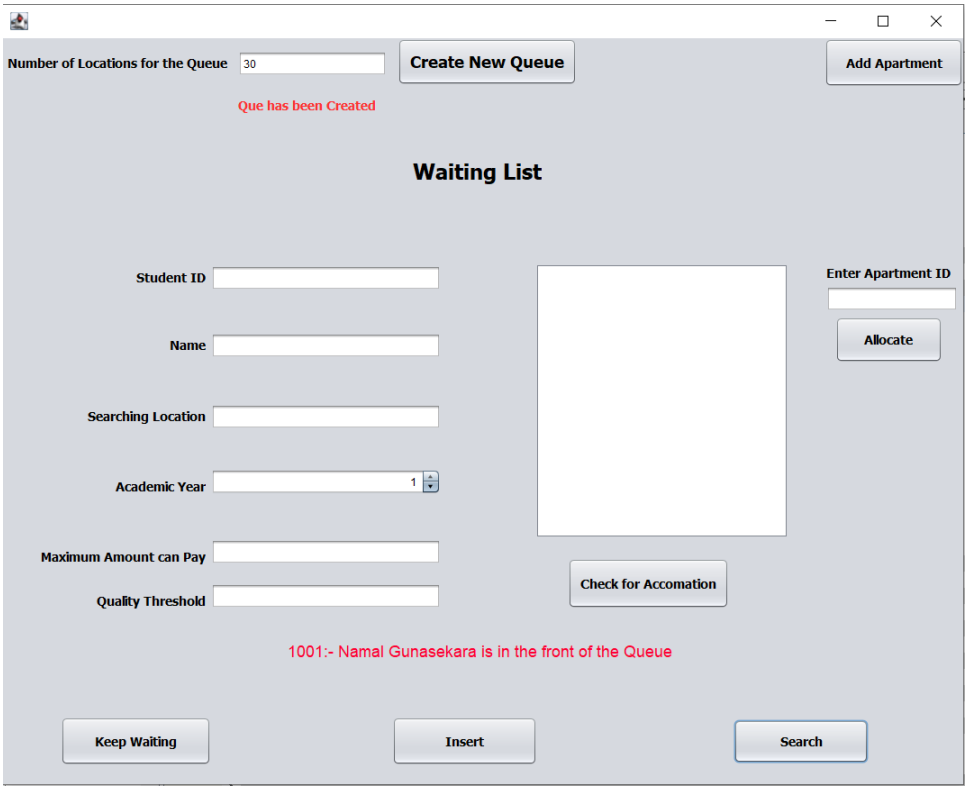
Student Housing System	
Test Cases	
Test Unit: Check for Accommodation	Tester: Tithira Withanaarachchi
Test Case ID: 03	Test Type: Black Box Testing
Description: Check for the Apartments which matches with the requirement of the Student.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date: : 17/06/2021
Title: Check for Apartments according to the Student Requirement.	Test Execution Time:

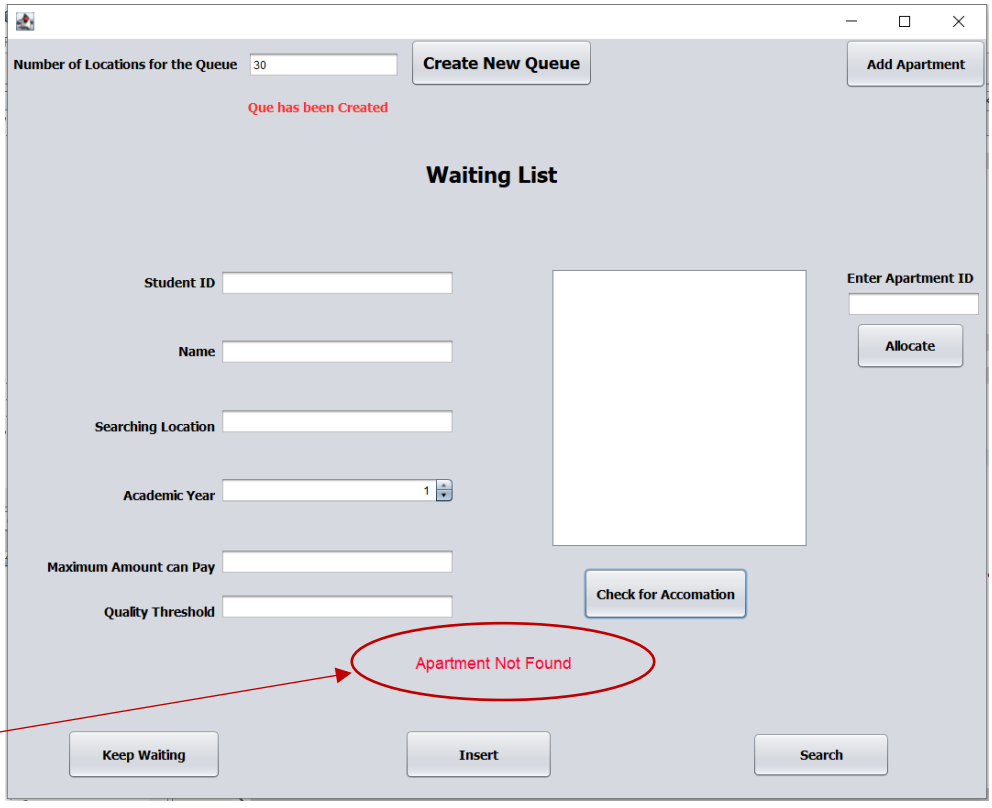
Step No	Test Steps	Test Data	Expected Results	Actual Result	Status(Pass/Fail)	Notes
01	Insert Apartments using the option Add Apartment. Note:- Add at least one Apartment which satisfies a Student in the Que.					
02	Create a Que for Waiting List.					
03	Insert Students to the Que.					
04	Search for name and the ID of the Student using "Search" option.					
05	Click on the "Check for Accommodation" Button.		List of Matching Apartments Displayed in the List panel.	Error Message Saying that "Apartment not Found", Though there are apartments which satisfies the Student Requirement.	Fail	
04	Click on the "Check for		List of Matching Apartments	List of Matching Apartments	Pass	

	Accommodation" Button.		Displayed in the List panel.	Displayed in the List panel.		

Scenario of the Error :-




Number of Locations for the Queue
30
Create New Queue
Add Apartment
Que has been Created
Waiting List
Student ID
Name
Searching Location
Academic Year
1
Maximum Amount can Pay
Quality Threshold
Enter Apartment ID
Allocate
Check for Accomation
1001:- Namal Gunasekara is in the front of the Queue
Keep Waiting
Insert
Search

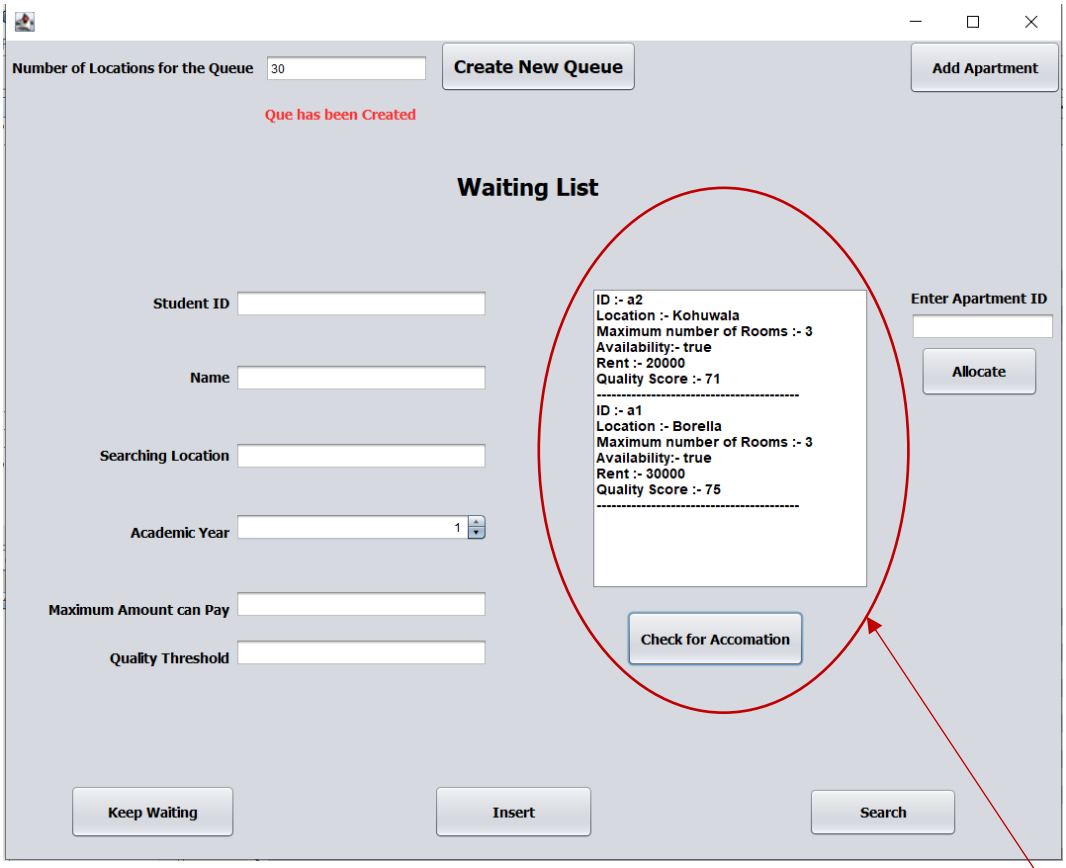

Number of Locations for the Queue
30
Create New Queue
Add Apartment
Que has been Created
Waiting List
Student ID
Name
Searching Location
Academic Year
1
Maximum Amount can Pay
Quality Threshold
Enter Apartment ID
Allocate
Check for Accomation
Apartment Not Found
Keep Waiting
Insert
Search

Error

The above error has been occurred due to not passing of the exact Apartment object from the Apartment form to the Waiting List form.

Successful Scenario:-

This error has been corrected by making the Apartment object created in the Apartment form, a Static object. Therefore, the same object created in Apartment object could be accessed in the waiting List form without creating a new object from the Apartment form.

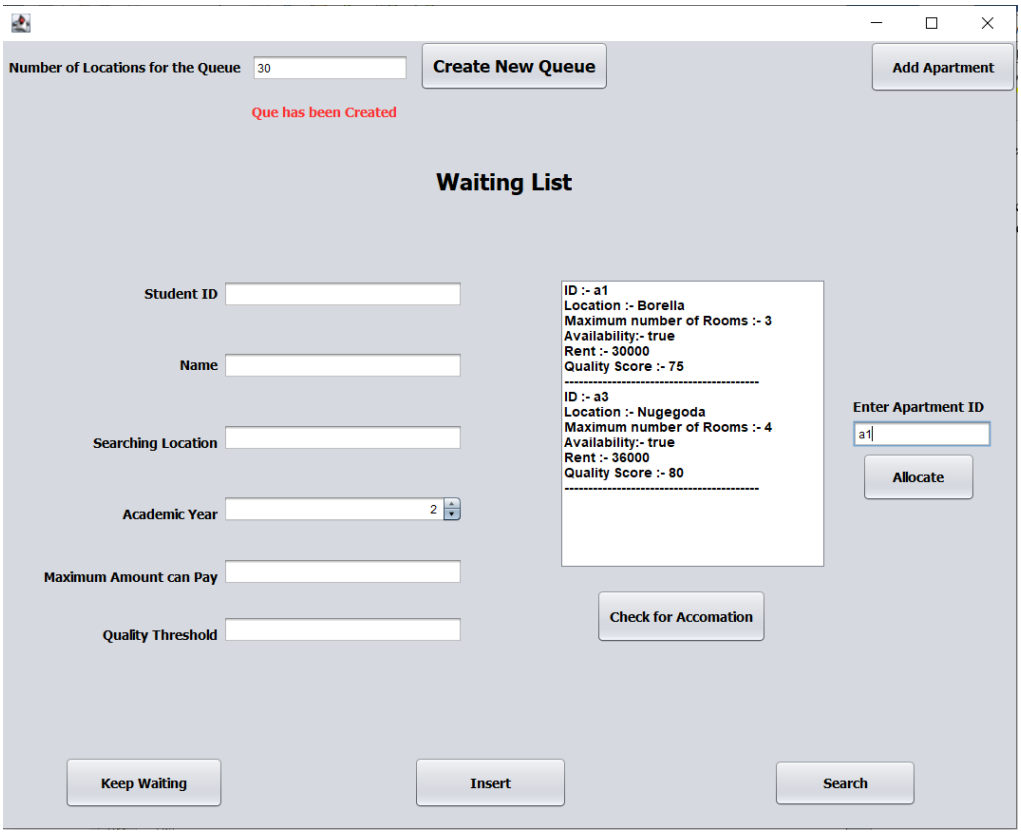


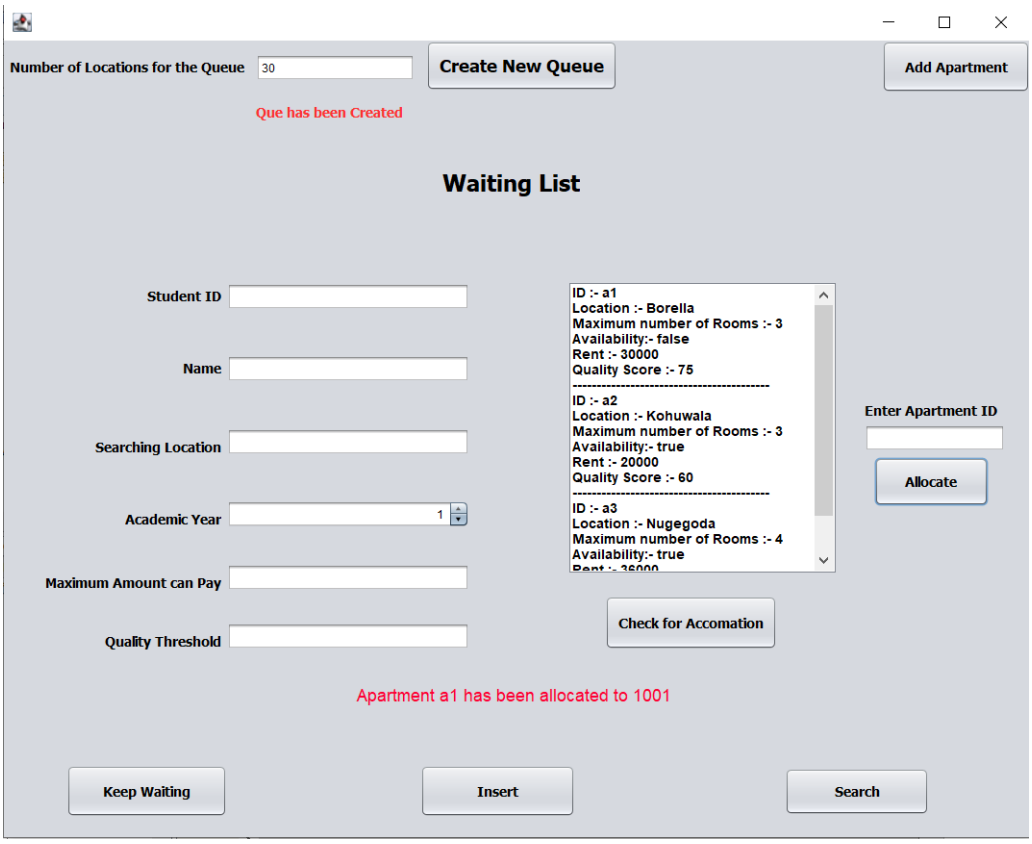
Matching Apartments to the Student at the front (Nimal Gunasekara).

Student Housing System	
Test Cases	
Test Unit: Allocate Apartment	Tester: Tithira Withanaarachchi
Test Case ID: 04	Test Type: Black Box Testing
Description: Enter the Apartment ID of the Apartment Selected by the Student and Allocate it to the Student.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date: : 17/06/2021
Title: Allocate apartment to Student in the Front using Apartment ID.	Test Execution Time:

Step No	Test Steps	Test Data	Expected Results	Actual Result	Status(Pass/Fail)	Notes
01	Insert Apartments using the option Add Apartment. Note:- Add at least one Apartment which satisfies a Student in the Que.					
02	Create a Que for Waiting List.					
03	Insert Students to the Que.					
04	Search for name and the ID of the Student using "Search" option.					
05	Click on the "Check for Accommodation" Button.					
06	Insert the ID of the Apartment accepted by the Student and Click on "Allocate" Button.	Apartment ID= a1	Displaying a Message "Apartment a1 has been allocated to <Student ID>" and the Availability of the Apartment that have	Displaying a Message "Apartment a1 has been allocated to <Student ID>" and the Availability of the Apartment that have	Pass	

			been Allocated to Student should be changed to “false” and displayed all apartments in the list panel.	been Allocated to Student should be changed to “false” and displayed all apartments in the list panel.		

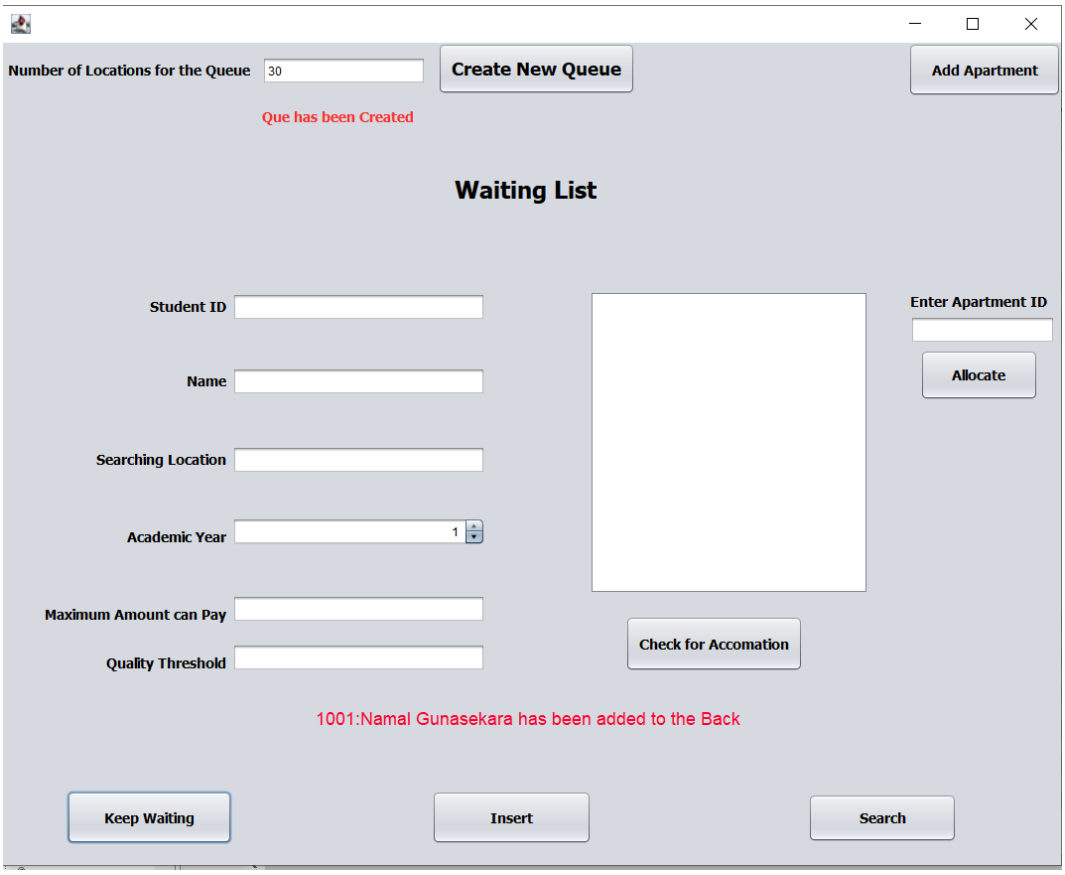




Student Housing System	
Test Cases	
Test Unit: Keep Student Waiting	Tester: Tithira Withanaarachchi
Test Case ID: 05	Test Type: Black Box Testing
Description: If the Student Rejects an Apartment, Student is added to the back of the Que.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date: : 17/06/2021
Title: Student is Added to the Back.	Test Execution Time:

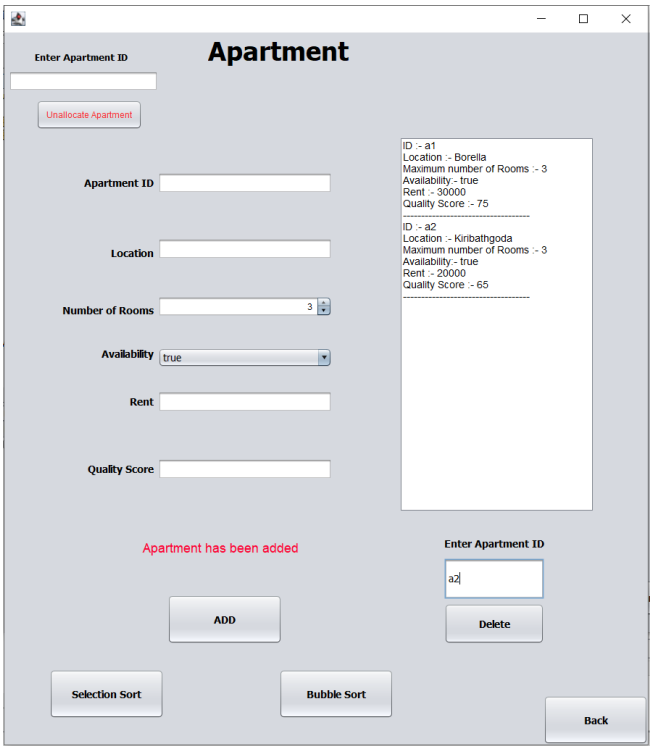
Step No	Test Steps	Test Data	Expected Results	Actual Result	Status(Pass/Fail)	Notes
02	Create a Que for Waiting List.					
03	Insert Students to the Que. Note:- Insert more than one Student					
04	Search for name and the ID of the					

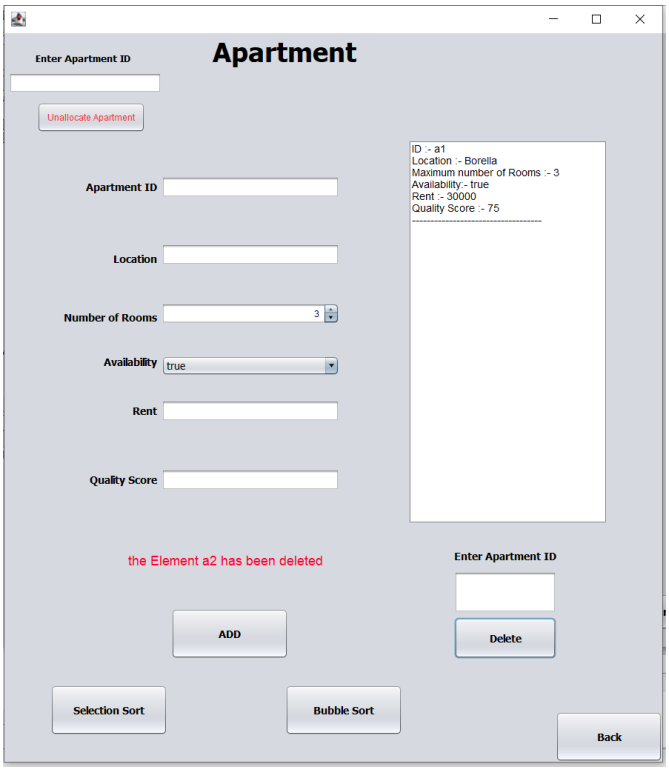
	Student using “Search” option.					
05	Click on the “Keep Waiting” Button.					
06	Click on the “Search” Button again		Display a message which has the structure “<id of the Student inserted 2nd to the Que>:- <Name of the Student Inserted 2nd to the Que> is in Front of the Que”	Display a message which has the structure “<id of the Student inserted 2nd to the Que>:- <Name of the Student Inserted 2nd to the Que> is in Front of the Que”	Pass	



Student Housing System	
Test Cases	
Test Unit: Delete an Apartment	Tester: Tithira Withanaarachchi
Test Case ID: 06	Test Type: Black Box Testing
Description: Delete an Apartment from the Apartment List if that Apartment is not available in the university contact pool anymore.	Test Executed by: Tithira Withanaarachchi
	Test Execution Date: : 17/06/2021
Title: Deleting an Apartment from the Apartment list.	Test Execution Time:

Step No	Test Steps	Test Data	Expected Results	Actual Result	Status(Pass/Fail)	Notes
01	Click on the "Add Apartment" option.					
02	Insert Apartments. Note:- Insert more than one Apartment.					
03	Put the Relevant ID of the Apartment that is needed to be deleted and Click on the "Delete" Button.	Apartment ID:- a2	Display a Message "the Element a2 has been added"	Display a Message "the Element a2 has been added"	Pass	


Apartment
Enter Apartment ID
Unallocate Apartment
Apartment ID
Location
Number of Rooms
3
Availability
true
Rent
Quality Score
ID :- a1
Location :- Borella
Maximum number of Rooms :- 3
Availability:- true
Rent :- 30000
Quality Score :- 75
ID :- a2
Location :- Kiribathgoda
Maximum number of Rooms :- 3
Availability:- true
Rent :- 20000
Quality Score :- 65
Apartment has been added
Enter Apartment ID
a2
ADD
Delete
Selection Sort
Bubble Sort
Back


Apartment
Enter Apartment ID
Unallocate Apartment
Apartment ID
Location
Number of Rooms
3
Availability
true
Rent
Quality Score
ID :- a1
Location :- Borella
Maximum number of Rooms :- 3
Availability:- true
Rent :- 30000
Quality Score :- 75
the Element a2 has been deleted
Enter Apartment ID
ADD
Delete
Selection Sort
Bubble Sort
Back

Task 3

Bubble Sort.

```
public class BubbleSort {

    public  static void bubbleSort(ArrayList<Apartment> arr){
        int n= arr.size();
        for(int i=0;i<n;i++){
            for(int j=1;j<(n-i);j++){
                if(arr.get(j-1).getRent()>arr.get(j).getRent()){
                    //swapping the elements
                    Collections.swap(arr, j-1, j);

                }
            }
        }
    }
}
```

Shown above is the code for bubble sort , which is used to sort the apartments in the increment order of their rental. There are 2 loops and “if ” conditional statement used inside the method. Outermost loop is used for traversing through the array and select the element with the maximum value using the inner loop to put it in the right-hand side corner.

Since an Array List is used for storing details about the Apartments, inbuilt swap() method is used for interchange the location of 2 Apartments if Apartment with a low rent is found in the right- hand side of the Array List.

Selection Sort

```
*/
public class SelectionSort {

    public static void selectionSort(ArrayList<Apartment> arr){
        for(int i=0;i<arr.size()-1;i++){
            int min=i;
            for(int j=i+1;j<arr.size();j++){
                if(arr.get(j).getRent()<arr.get(min).getRent()){
                    min=j;
                }

            }
             Collections.swap(arr, min, i);
        }
    }
}
```

Selection sort is implemented in the above code using 2 loops and a conditional statement. Outer loop is used to traverse through the Array List while using the inner loop to compare the proceeding elements with the minimum value considered initially and finally find the actual minimum value from the array continuously and put the minimum values to the left-hand side of the Array List in the increment order.

Inbuilt swap method of Array List is used in the above situation as well to swap the locations of two elements if a lower value than the minimum value selected is found within the proceeding elements.

Performance Comparison of Bubble Sort and Selection Sort

Basis for Comparison	**Bubble Sort**	**Selection Sort**
General	Adjacent elements are compared and swapped if the next element is smaller than the element at the current location.(for increment Order)	Largest/Minimum value is selected and placed in the right-hand side/left-hand side corner of the data structure.(for increment Order)
Best Case time Complexity	O(n)	$O(n^2)$
Efficiency	Comparatively Inefficient	Comparatively Efficient
Stability	Comparatively Stable	Comparatively not Stable
Method used	Swapping	Selection
Speed	Comparatively Slow	Comparatively Fast

Table 1 Table for Performance Comparison of the Bubble Sort and Selection Sort [1]

Best Case time Complexity:-

According to above scenario, the number of Apartments in the University Contact pool is 100.

Therefore, the Best Case Time Complexity of sorting by using Bubble Sort,

Best case Time Complexity=O(n)=100

Therefore, the Best Case Time Complexity of sorting by using Selection Sort,

Best case Time Complexity=$O(n^2)$=10,000

Method used for Sorting:-

Bubble Sort:

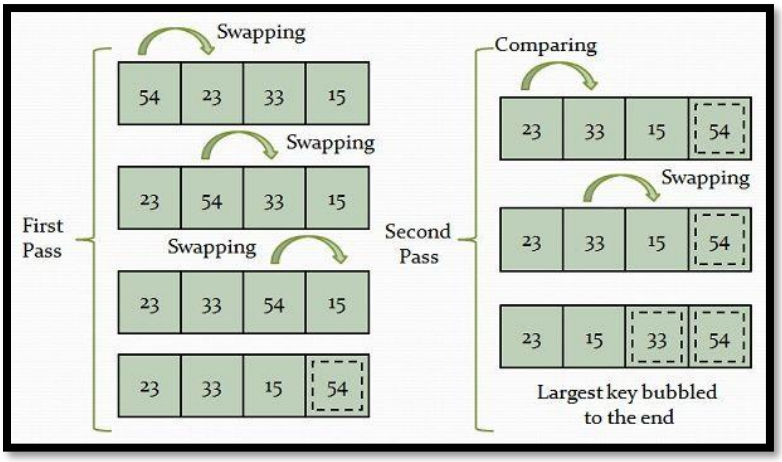


Figure 1 Illustration of the Method used in Bubble Sort [1]

As shown above, in bubble sort, it will swap the adjacent elements if it finds a element with larger value in the right hand side of the element at the current location., otherwise it will move to the element in the next location without swapping. Ultimately, element with maximum value will be brought to the right-hand side corner of the data structure.

Selection Sort:

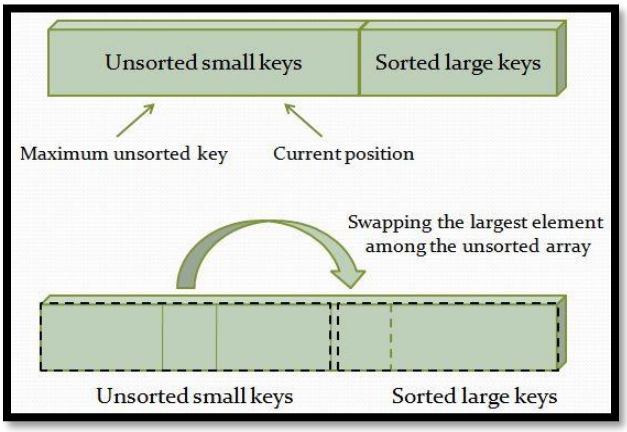


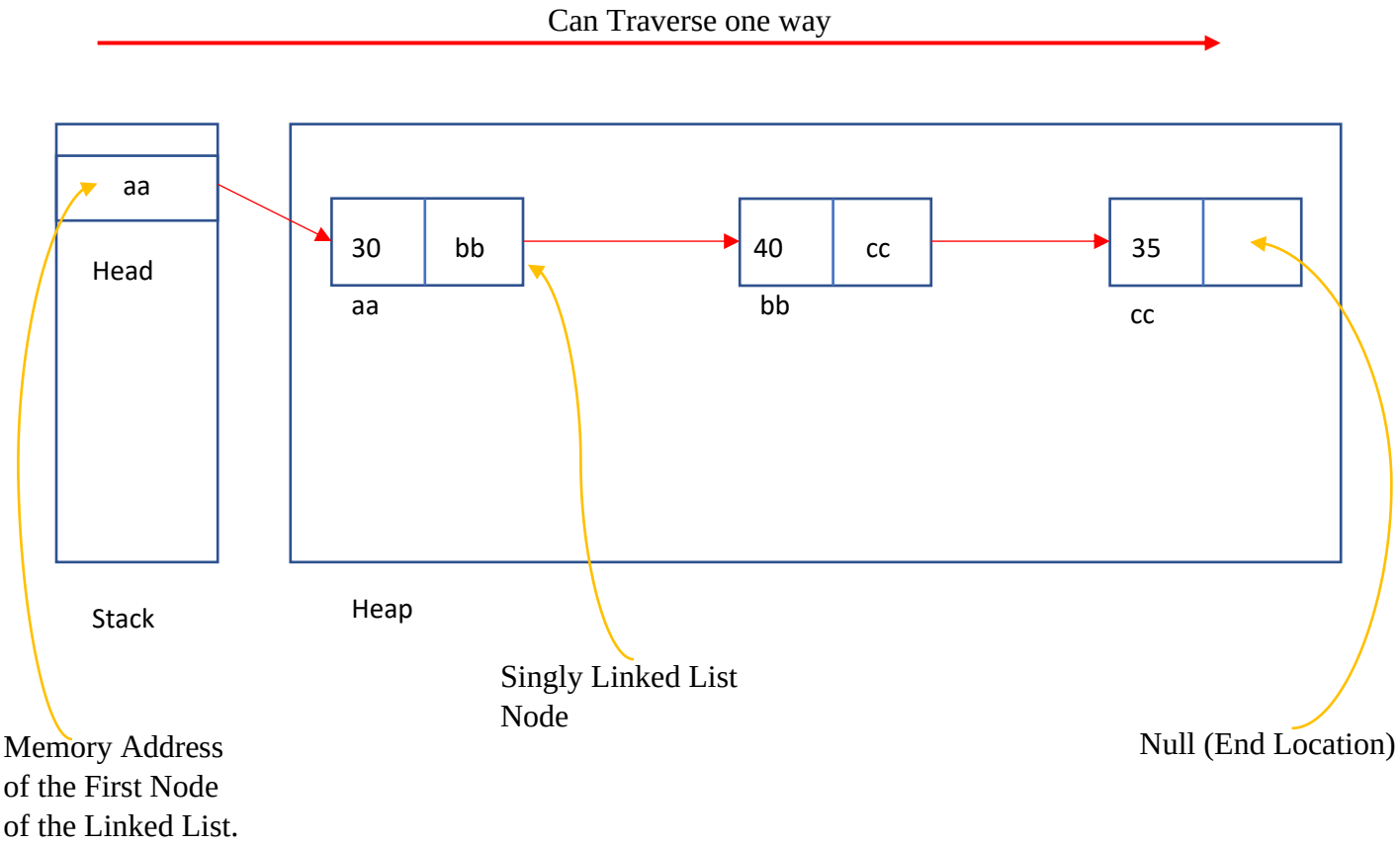
Figure 2 illustration of the sorting method used in Selection sort.

As shown above, the element with maximum/minimum value will be selected by running through the array using a loop and it will be swapped with the last element in the unsorted part of the array in Selection Sort.

Task 4

Singly Linked List-

Singly Linked List is a Dynamic Data Structure which has nodes with two sections in it, each for Storing the element and the other for storing the reference(memory address) for the next node. By having this reference to the next node, it creates a link between the Current Node and the next node. This make it possible to access the Whole set of linked nodes which is known as a Link List through one reference which is usually known as the head.



These Links between the nodes makes it possible to traverse through the list starting from head to the last node in Singly Linked List.

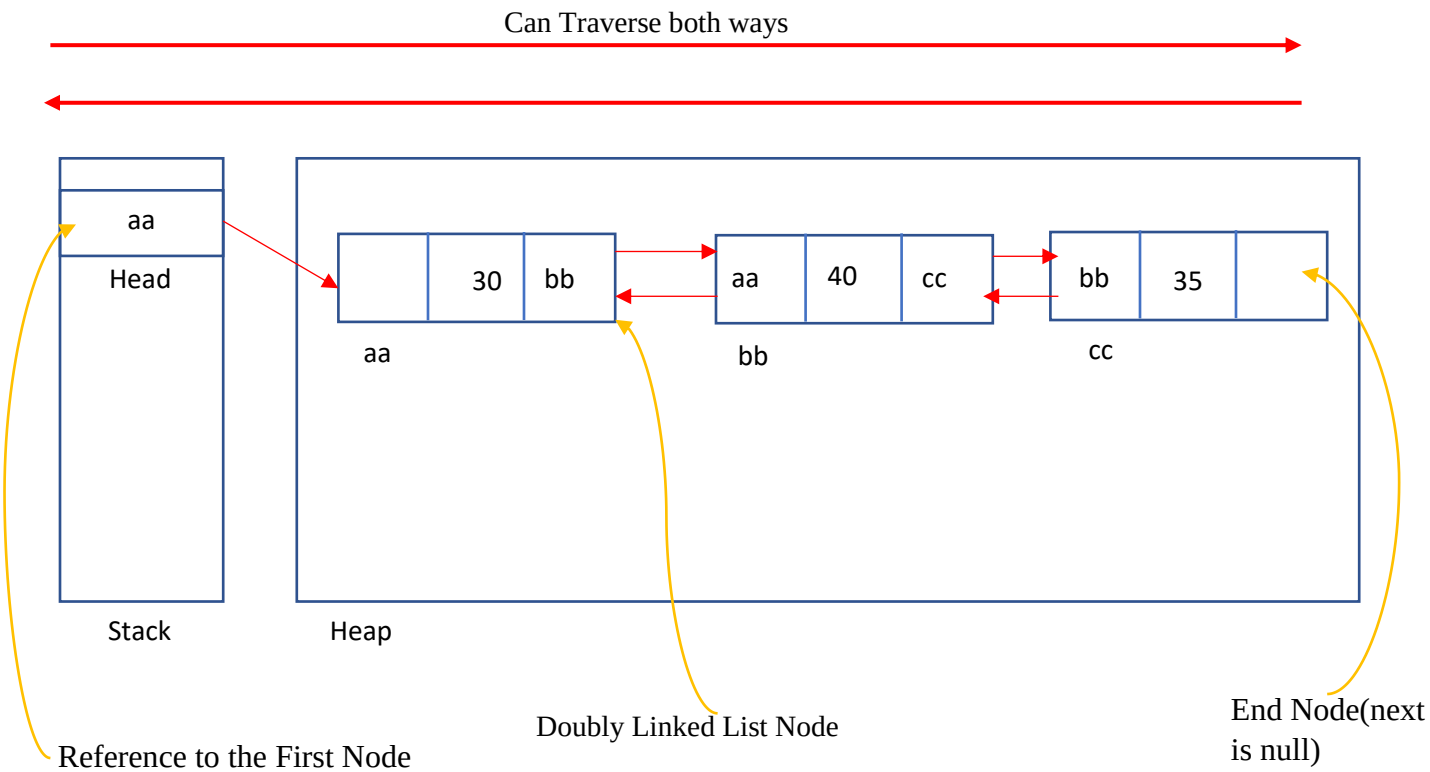
Some of the Operations that can be done in a Singly Linked List are,

- Add element to the back,
- Add element to any specified Location,
- Add element to front of the linked list,
- Delete from anywhere.
- Search any specified element,
- Display all the Elements.

Doubly Linked List-

Doubly Linked List is a Dynamic data Structure which contains nodes with 3 sections each for Storing the ,

- reference for the previous node,
- the Element,
- Reference to the next node.

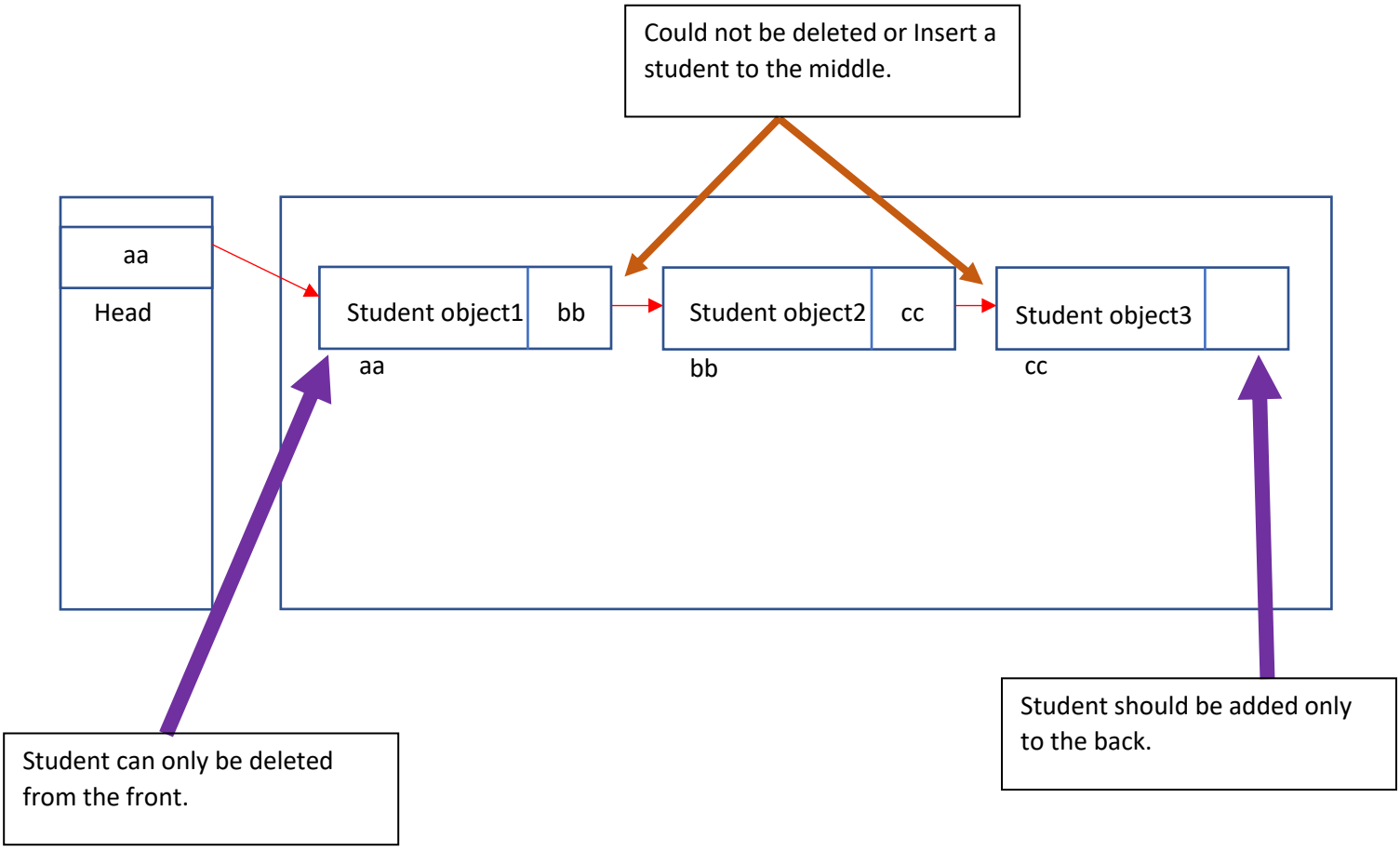


Doubly Link list has a similar structure to singly linked list, but it has a stronger link between each node due to the availability of an extra section for storing the reference for the previous node.

This reference to the previous node makes it possible to traverse in the opposite direction as well other than traversing in the direction of head to the end location.

Doubly linked list will perform all the operations done in the Singly Linked List.

Implementing the Scenario using the Linked List.



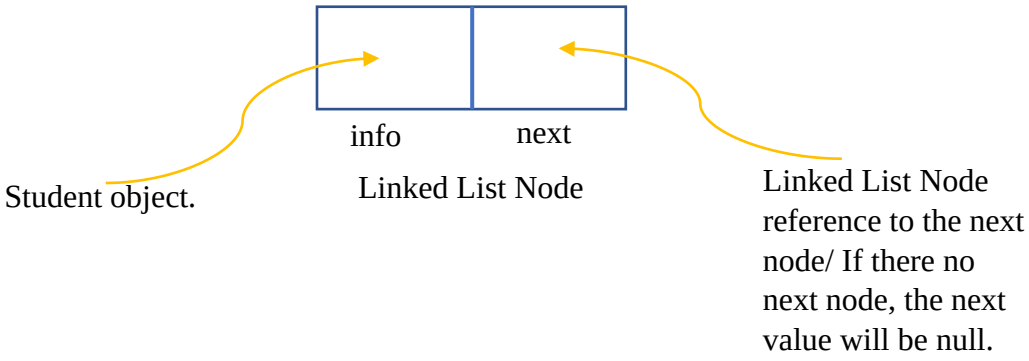
In the above scenario, most appropriate way to store the details regarding the students is in the form of a Que. Therefore, if we use a Linked List to implement the above scenario, we should not make all the possible operations available for the Linked List. Only the operations which can be performed in a que should be available in the Linked List.

Therefore, the possible operations that could be performed by the Linked List implemented for the above scenario would be,

- Can be Inserted Only to the Back,
- Student can be deleted only from the front,
- Searching could be done only for the student at the front.

```java
public class LinkListNode {
    public Student info;//Student Object reference
    public LinkListNode next;//Link List Node reference
    public LinkListNode(Student st){//Linked List Node Constructor
        info=st;
        next=null;
    }

}
```

Figure 3 Image of "LinkListNode.java" code



The above diagram represents the structure of the node created by the constructor of the Linked list class according to the code shown.

The type of data Structure, the Link List Belongs to:-

Static and Dynamic Data Structures.

Data Structures are used to Store data in an organized manner in order to use the memory and time efficiently when doing operations with data. As a whole it will also be used to reduce the complexity of the code as well [2].

There are 2 categories of data Structures. They are namely,

- Static Data Structures.
- Dynamic Data Structures.

Static Data Structure.

Static data structure is created in the compile time. The values stored inside the data structure can be updated throughout, but it's not possible to change the number of memory locations allocated to the data structure, after creating it.

Eg-Array.

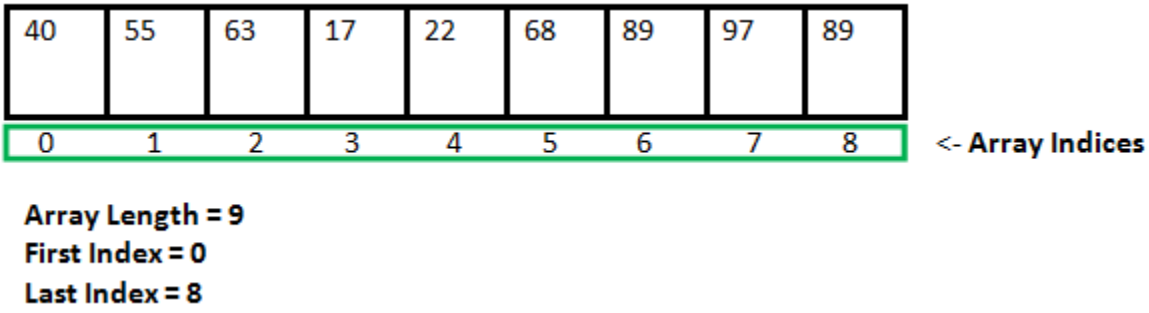


Figure 4 Diagram which illustrate the structure of a Static Data Structure- Array [2].

Dynamic Data Structure

Dynamic data structure is created in the runtime. The memory locations allocated to a dynamic data structure can be modified while doing operations using it.

Eg- Array List, Linked List

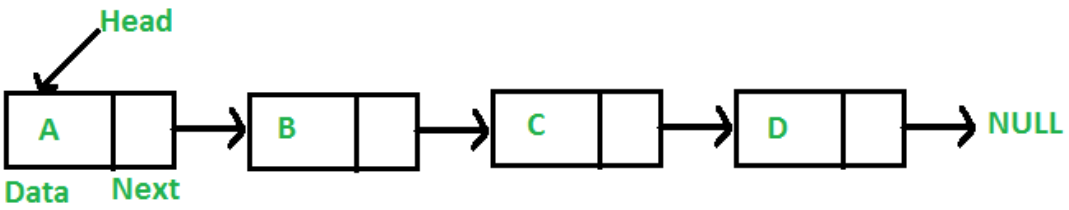


Figure 5 Diagram which illustrate the structure of a Dynamic Data Structure- Singly Linked List [2].

Static Data Structures	**Dynamic Data Structures**
Easier access to elements.	Accessing elements is comparatively harder.
Created in compile time	Created in runtime
Allocated locations are not flexible	Allocated locations are flexible.

Conclusion :-

Linked List is a data Structure which will create or delete nodes each time we enter or delete an element. As a result, number of locations allocated to it will be changed according to the requirement.

Therefore, as explained above Linked list is referred to as a Dynamic Data Structure because of it's flexibility to modify the number of locations allocated to it.

References

[1] "Tech Differences," [Online]. Available: https://techdifferences.com/difference-between-bubble-sort-and-selection-sort.html. [Accessed 18 06 2021].

[2] "Geeks for Geeks," [Online]. Available: https://www.geeksforgeeks.org/static-data-structure-vs-dynamic-data-structure/. [Accessed 17 06 2021].