JOBSHEET XI
QUEUE

1. Learning Objectives

After completing this practical session, students will be able to:

1. Understand the structure and concept of the Queue data structure
2. Create and declare a Queue data structure
3. Implement Queue algorithms using arrays

2. Labs 1

2.1 Experiment 1 : Queue Basic Operations

Allocated Time : 90 minutes

In this experiment, we will implement the basic operations of the Queue algorithm. Take a look at the following Queue class diagram:

Queue
data: int[] front: int rear: int size: int max: int
Queue(n: int) isFull(): boolean isEmpty(): boolean enqueue(dt: int): void dequeue(): int peek: void print(): void clear(): void

The following steps will implement Java program from the above design.

2.1.1. Experiment Steps

1. Create a new folder named **P1Jobsheet11** or another prefered name, inside the Praktikum-ASD repository, then create a new class named **Queue**.
2. Add the attributes of the **Queue** as shown in the class diagram, then add the constructor as described below:

```
int[] data;
int front, rear, size, max;

public Queue(int n){
   max = n;
```

```
    data = new int[max];
    size = 0;
    front = rear = -1;
}
```

3. Create a boolean method named **isEmpty** to check whether the queue is empty.

```
boolean isEmpty(){
        if(size == 0){
            return true;
        }else{
            return false;
        }
    }
```

4. Create a boolean method named **isFull** to check whether the queue is full.

```
boolean isFull(){
        if(size == max){
            return true;
        }else{
            return false;
        }
    }
```

5. Create a void method named **peek** to display the element at the **front** of the queue.

```
void peek(){
        if(!isEmpty()){
            System.out.println("Front data: "+data[front]);
        }else{
            System.out.println("Queue is empty!!!");
        }
    }
```

6. Create a void method named **print** to display all elements in the queue, from the **front** to the **rear** position.

```
void print(){
        if(!isEmpty()){
            int i = front;
            while(i != rear){
                System.out.print(data[i]+" ");
                i = (i+1)%max;
            }
            System.out.println(data[i]+" ");
            System.out.println("Number of element: "+size);
        }else{
```

```
            System.out.println("Queue is empty!!!");
        }
    }
```

7. Create a void method named **clear** to remove all elements from the queue.

```
void clear(){
        if(!isEmpty()){
            front = rear = -1;
            size = 0;
            System.out.println("All data has been successfully
removed!");
        }else{
            System.out.println("Queue is already empty!!!");
        }
    }
```

8. Create a void method named **enqueue** to add an element to the queue, using a parameter **dt** of type **int**.

```
void enqueue(int dt){
        if(!isFull()){
            if(isEmpty()){
                front=rear=0;
            }else{
                if(rear==max-1){
                    rear=0;
                }else{
                    rear++;
                }
            }
            data[rear]=dt;
            size++;
            System.out.printf("%d is successfully added at index %d\n",
dt, rear);
        }else{
            System.out.println("Queue is full!!!");
        }
    }
```

9. Create an **int** method named **dequeue** to remove and return the data from the **front** position of the queue.

```
int dequeue(){
        int dt = 0;
        if(!isEmpty()){
            dt = data[front];
            size--;
```

```
            if(isEmpty()){
                front=rear=-1;
            }else{
                if(front==max-1){
                    front=0;
                }else{
                    front++;
                }
            }
        }else{
            System.out.println("Queue is empty!!!");
        }
        return dt;
    }
```

10. Next, create a new class named **QueueMain**. Create a **void** method named **menu** to display and handle the program's menu options during execution.

```
static void menu(){
        System.out.println("Available menu:");
        System.out.println("1. Enqueue");
        System.out.println("2. Dequeue");
        System.out.println("3. Print");
        System.out.println("4. Peek");
        System.out.println("5. Clear");
        System.out.println("=======================");
    }
```

11. Create the **main** method, then declare a **Scanner** object named **sc**.
12. Create a variable named **n** to store the user input representing the maximum number of elements that can be stored in the queue.

```
Scanner sc = new Scanner(System.in);
System.out.print("Input maximum number of data: ");
int n = sc.nextInt();
```

13. Instantiate a **Queue** object named **Q**, passing **n** as the queue's element capacity.

```
Queue Q = new Queue(n);
```

14. Declare an integer variable named **choice** to store the user's menu selection.
15. Use a **do-while** loop to keep the program running based on user input. Inside the loop, use a **switch-case** statement to execute queue operations according to the user's menu selection.

```
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Input maximum number of data: ");
        int n = sc.nextInt();
        Queue Q = new Queue(n);
```

```
        int choice = -1;
        do {
            menu();
            choice = sc.nextInt();
            switch (choice) {
                case 1:
                    System.out.print("Input new data: ");
                    int newData = sc.nextInt();
                    Q.enqueue(newData);
                    break;
                case 2:
                    int outData = Q.dequeue();
                    if (outData != 0) {
                        System.out.println("Removed data: " + outData);
                        break;
                    }
                case 3:
                    Q.print();
                    break;
                case 4:
                    Q.peek();
                    break;
                case 5:
                    Q.clear();
                    break;
            }
        } while (choice==1 || choice==2 || choice==3 || choice==4 ||
choice==5);
    }
```

16. Compile and run the **QueueMain** class, then observe the results.

2.1.2. Output Verification

Verify the output with the following:

```
Input maximum number of data: 4
Available menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
=======================
1
Input new data: 15
15 is successfully added at index 0
Available menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
```

```
======================
1
Input new data: 31
31 is successfully added at index 1
Available menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
======================
3
15 31
Number of element: 2
Available menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
======================
4
Front data: 15
Available menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
======================
2
Removed data: 15
Available menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
======================
3
31
Number of element: 1
Available menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
======================
```

2.1.3. Question

1. In the constructor, why are the initial values of the **front** and **rear** attributes set to **-1**, while the size attribute is set to **0**?
2. In the **Enqueue** method, explain the meaning and purpose of the following code snippet!

```
if (rear == max - 1) {
    rear = 0;
```

3. In the **Dequeue** method, explain the meaning and purpose of the following code snippet!

```
if (front == max - 1) {
    front = 0;
```

4. In the **print** method, why does the loop variable **i** start from **front** instead of **0** (i.e., int i = 0)?
5. Review the **print** method again, and explain the meaning of the following code snippet!

```
i = (i + 1) % max;
```

6. Show the code snippet that represents a **queue overflow**!
7. When a queue overflow or underflow occurs, the program continues to run and only displays informational text. Modify the program so that it stops when a queue overflow or underflow happens!

2.2. Experiment 2 : Academic Service Queue

Allocated Time : 90 minutes

In this experiment, we will create a program that illustrates the service provided by the Academic Admin. Please refer to the following class diagram:

Student
nim:String name: String studyProgram: String className: String
Student (nim: String, name: String, studyProgram: String, className: String) void print()

2.2.1. Experiment Steps

Based on the class diagram, a **Student** class will be created in Java.

1. Create a new class named **Student**
2. Add the attributes as shown in the class diagram. Then, create the parametric constuctor as follows:

```
public Student(String nim, String name, String studyProgram, String
className){
        this.nim = nim;
        this.name = name;
        this.studyProgram = studyProgram;
        this.className = className;
    }
```

Provide **print()** method to display the whole data of student.

```
void print(){
        System.out.println(nim+" - "+name+" - "+studyProgram+" -
"+className);
    }
```

3. Copy the **Queue** class code from Experiment 1 to be reused in this experiment, and rename the class to **StudentQueue**. Since in Experiment 1, the data stored in the queue was only an **integer** array, while in this experiment the data used is Student object, you will need to modify the **StudentQueue** class accordingly.

```
Student[] data;
int front, rear, size, max;

public StudentQueue(int n){
        max = n;
        data = new Student[max];
        size = 0;
        front = 0;
        rear = -1;
}
```

4. Modify the **StudentQueue** class by changing the type of the data array from int[] to **Student**[], since in this case the data to be stored consists of **Student** objects. The modifications should be applied to the attributes, the **enqueue** method, and the **dequeue** method.

```
void enqueue(Student dt){
        if(isFull()){
            System.out.println("Queue is full!!!");
            return;
        }
        rear = (rear+1)%max;
        data[rear] = dt;
        size++;
        System.out.printf("%s is successfully added at index %d\n",
dt.name, rear);
}

Student dequeue(){
        if(isEmpty()){
            System.out.println("Queue is empty!!!");
            return null;
        }
        Student dt = data[front];
        front = (front+1)%max;
        size--;
        return dt;
}
```

5. Next, implement the **peek** and **print** methods to display the front-most data in the service queue and to display all the data in the service queue, respectively.

```
void peek(){
```

```
        if(!isEmpty()){
            System.out.println("Front data: ");
            data[front].print();
        }else{
            System.out.println("Queue is empty!!!");
        }
    }

    void print(){
        if(isEmpty()){
            System.out.println("Queue is empty!!!");
            return;
        }
        int i = front;
        while(i != rear){
            data[i].print();
            i = (i+1)%max;
        }
        data[i].print();
        System.out.println("Number of element: "+size);
    }
```

6. Next, create a new class named **StudentQueueMain** within the same package. In this class, define the **main** function and declare a **Scanner** object with the name **sc**.
7. Then, instantiate an **StudentQueue** object named **queue** with the parameter set to the maximum queue size (for example, 5).
8. Declare an integer variable named **choice** to store the user's menu selection.
9. Add the following code to loop through the menu options based on the input provided by the user:

```
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        StudentQueue queue = new StudentQueue(5);
        int choice;
        do {
            System.out.println("\n=== Academic Service Menu ===");
            System.out.println("1. Enqueue Student");
            System.out.println("2. Dequeue Student");
            System.out.println("3. Display Front");
            System.out.println("4. Display All Data");
            System.out.println("5. Queue Size");
            System.out.println("0. Exit");
            System.out.print("Choose a menu: ");
            choice = sc.nextInt();
            sc.nextLine();
```

```
            switch (choice) {
                case 1:
                    System.out.print("NIM   : ");
                    String nim = sc.nextLine();
                    System.out.print("Name  : ");
                    String name = sc.nextLine();
                    System.out.print("Study Program : ");
                    String studyProgram = sc.nextLine();
                    System.out.print("Class : ");
                    String className = sc.nextLine();
                    Student std = new Student(nim, name, studyProgram,
classname);
                    queue.enqueue(std);
                    break;
                case 2:
                    Student studentBeingServed = queue.dequeue();
                    if (studentBeingServed != null) {
                        System.out.println("Student being served:");
                        studentBeingServed.print();
                    }
                    break;
                case 3:
                    queue.peek();
                    break;
                case 4:
                    queue.print();
                    break;
                case 5:
                    System.out.println("Queue size: " + queue.size);
                    break;
                case 0:
                    System.out.println("Thanks!!");
                    break;
                default:
                    System.out.println("Invalid menu!!");
            }
        } while (choice != 0);
    }
```

10. Compile and run **StudentQueueMain** class and observe the result!

2.2.2 Result Verification

Verify the output with the following:

```
=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
```

```
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 1
NIM   : 111
Name  : Joseph
Study Program : TI
Class : 1I
Joseph is successfully added at index 0

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 1
NIM   : 222
Name  : Brian
Study Program : TI
Class : 1I
Brian is successfully added at index 1

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 1
NIM   : 333
Name  : Herco
Study Program : TI
Class : 1I
Herco is successfully added at index 2

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 1
NIM   : 444
Name  : Bima
Study Program : TI
Class : 1I
Bima is successfully added at index 3

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 1
NIM   : 555
Name  : Kinan
Study Program : TI
Class : 1I
Kinan is successfully added at index 4

=== Academic Service Menu ===
```

```
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 1
NIM   : 6666
Name  : Nahda
Study Program : TI
Class : 1I
Queue is full!!!

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 5
Queue size: 5

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 4
111 - Joseph - TI - 1I
222 - Brian - TI - 1I
333 - Herco - TI - 1I
444 - Bima - TI - 1I
555 - Kinan - TI - 1I
Number of element: 5

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 3
Front data:
111 - Joseph - TI - 1I

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 2
Student being served:
111 - Joseph - TI - 1I

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
```

```
Choose a menu: 5
Queue size: 4

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu: 4
222 - Brian - TI - 1I
333 - Herco - TI - 1I
444 - Bima - TI - 1I
555 - Kinan - TI - 1I
Number of element: 4

=== Academic Service Menu ===
1. Enqueue Student
2. Dequeue Student
3. Display Front
4. Display All Data
5. Queue Size
0. Exit
Choose a menu:
```

2.2.3 Questions

1. What is the main difference between the **Queue** (experiment 1) and **StudentQueue** classes in terms of the **data type** they manage? Why is this change in data type important?
2. How do the **enqueue** and **dequeue** methods in **StudentQueue** differ from those in **Queue**?
3. Examine the **constructor** of the **StudentQueue** class: why is the **front** attribute initialized to 0, whereas in the **Queue** class from Experiment 1, the **front** attribute is initialized to -1?
4. Modify the program by adding a new method called **viewRear()** in the **StudentQueue** class to check the queue element at the **rear** position. Also, update the menu in the **StudentQueueMain** class by adding option 6: “Check rear of the queue,” so that the **viewRear()** method can be invoked.

2.3 Assignment

Time : 120 minutes

Create a queue-based program to simulate the approval queue for Student Course Registration Forms (KRS) by Academic Advisors (DPA). When a student joins the queue, they must first register their personal data (as in Experiment 2). Use queue classes similar to those used in Experiment1 and 2, and implement the following methods:

- Check if the queue is **empty**, check if the queue is **full**, and **clear** the queue.
- **Add** a student to the queue (enqueue), and process the KRS approval (dequeue) – each approval session processes **2 students at a time** (from the first two in the queue).

- Show **all students** in the queue, show the **first two** students in line, and show the **last student** in the queue.
- Print the **total number of students** in the queue, print the **number of students who have completed the KRS approval process**.
- The **maximum queue size is 10**, and each DPA can handle up to **30 students**. Display the number of students who have not yet completed the KRS approval process.

Draw the **class diagram** for the queue structure and implement all the required methods through a **menu-based interface** in the main function.