Data Structure and Algorithm Practicum
Queue

Name
Muhammad Baihaqi Aulia Asy'ari

NIM
2241720145

Class
1I

Department
Information Technology

Study Program
D4 Informatics Engineering

1.1 Learning Objective

After finishing this topic, students must be able to:

1. Understand the basic concept of Queue
2. Understand the basic operation of Queue
3. Implement the Queue concept as well as the operation in a program by using Java

1.2 Lab Activities 1

1.2.1 Steps

1. We will create a program based on this following class diagram

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

2. Create a new project named **Jobsheet8**, create a new package with name **practicum1**. After that, create a new class **Queue**
3. Add max, size, front,rear and Q as its attributes based on the class diagram above.
4. Add a constructor with parameter and method create as illustration below.

```
public Queue(int n) {
    max = 4;
    Create();
}
```

Within the constructor, there is a code to execute **create()**. then we make the create function

```
public void Create() {
    Q = new int[max];
    size = 0;
    front = rear = -1;
}
```

5. Create a method isEmpty with boolean as its return data type. We use this function to identify whether a queue is empty or not

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

6. Create a method isFull with boolean as its return data type. We use this function to identify whether a queue is full or not

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

7. Create **peek()** with void as its return type to display the data at the beginning of the queue

```
public void peek() {
    if (!isEmpty()) {
        System.out.println("The first element : " + Q[front]);
    } else {
        System.out.println("Queue is still empty");
    }
}
```

8. Create **print()** with void as its return type to display the data from the beginning until the end of the queue

```
public void print() {
    if (!isEmpty()) {
        System.out.println("Queue is still empty");
    } else {
        int i = front;
        while (i != rear) {
            System.out.println(Q[i] + " ");
            i = (i+1) % max;
        }
        System.out.println(Q[i] + " ");
        System.out.println("Element amount : " + size);
    }
}
```

9. Create **clear()** with void as its return type to remove all data in a queue

```
public void clear() {
    if (!isEmpty()) {
        front = rear = -1;
        size = 0;
        System.out.println("Queue has been cleared
        ↪  successfully");
    } else {
        System.out.println("Queue is still empty");
    }
}
```

10. . Next up, we make a new method **enqueue()** to insert a new data in a queue that has integer datatype as its parameter

```
public void enqueue(int data) {
    if (isFull()) {
        System.out.println("Queue is already full");
    } else {
        if (isEmpty()) {
            front = rear = 0;
        } else {
            if (rear == max - 1) {
                rear = 0;
            } else {
                rear++;
            }
        }
        Q[rear] = data;
```

```
            size++;
        }
    }
```

11. . Create method **dequeue()** with integer as its return type. We will need this function whenever we want to remove the last data inside the queue

```
public int dequeue() {
    int data = 0;
    if (isEmpty()) {
        System.out.println("Queue is still empty");
    } else {
        data = Q[front];
        size--;
        if (isEmpty()) {
            front = rear = -1;
        } else {
            if (front == max - 1) {
                front = 0;
            } else {
                front++;
            }
        }
    }
    return data;
}
```

12. . Next, we create a new class named **QueueMain** still at the same package **Practicum1**. To create a menu with void return type to allow user choose which function to be executed when the program runs

```
public static void menu() {
    System.out.println("Choose 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("================================");
}
```

13. . Create a main function, and declare the Scanner object with name **sc**

14. . Create **n** variable to store input of how many elements that can be stored within the queue

```
Scanner sc = new Scanner(System.in);
System.out.print("Insert maximum queue : ");
int n = sc.nextInt();
```

15. . Instantiate the Queue object with name **Q** and set the parameter **n** as its queue length

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

16. . Declare the input of menu selected by user

17. . Loop with do-while to run the program based on the given input. Inside the loop, there is a switch case condition to manipulate queue based on the given input

```
int choose;
do {
    menu();
    choose = sc.nextInt();
    switch (choose) {
        case 1:
            System.out.print("Insert new data: ");
            int newData = sc.nextInt();
            Q.enqueue(newData);
            break;

        case 2:
            int removeData = Q.dequeue();
            if (removeData != 0) {
                System.out.println("Data removed : " +
                ↪  removeData);
                break;
            }

        case 3 :
            Q.print();
            break;

        case 4:
            Q.peek();
            break;

        case 5:
            Q.clear();
```

```
                break;
        }
    } while (choose <= 5 && choose >= 1);
```

18. . Compile the program and run the **QueueMain** class. And observe the result

```
package practicum1;

public class Queue {
    int max, size, front, rear;
    int[] Q;

    public Queue(int n) {
        max = n;
        create();
    }

    public void create() {
        Q = new int[max];
        size = 0;
        front = rear = -1;
    }

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

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

    public void peek() {
        if (!isEmpty()) {
            System.out.println("The first element : " +
            ↪  Q[front]);
```

```
        } else {
            System.out.println("Queue is still empty");
        }
    }

    public void print() {
        if (!isEmpty()) {
            System.out.println("Queue is still empty");
        } else {
            int i = front;
            while (i != rear) {
                System.out.println(Q[i] + " ");
                i = (i+1) % max;
            }
            System.out.println(Q[i] + " ");
            System.out.println("Element amount : " + size);
        }
    }

    public void clear() {
        if (!isEmpty()) {
            front = rear = -1;
            size = 0;
            System.out.println("Queue has been cleared
            ↪  successfully");
        } else {
            System.out.println("Queue is still empty");
        }
    }

    public void enqueue(int data) {
        if (isFull()) {
            System.out.println("Queue is already full");
        } else {
            if (isEmpty()) {
                front = rear = 0;
            } else {
                if (rear == max - 1) {
                    rear = 0;
                } else {
                    rear++;
                }
```

```
            }
            Q[rear] = data;
            size++;
        }
    }

    public int dequeue() {
        int data = 0;
        if (isEmpty()) {
            System.out.println("Queue is still empty");
        } else {
            data = Q[front];
            size--;
            if (isEmpty()) {
                front = rear = -1;
            } else {
                if (front == max - 1) {
                    front = 0;
                } else {
                    front++;
                }
            }
        }
        return data;
    }
}
```

```
package practicum1;

import java.util.Scanner;

public class QueueMain {
    public static void menu() {
        System.out.println("Choose 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("================================");
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Insert maximum queue : ");
        int n = sc.nextInt();

        Queue Q = new Queue(n);

        int choose;
        do {
            menu();
            choose = sc.nextInt();
            switch (choose) {
                case 1:
                    System.out.print("Insert new data: ");
                    int newData = sc.nextInt();
                    Q.enqueue(newData);
                    break;

                case 2:
                    int removeData = Q.dequeue();
                    if (removeData != 0) {
                        System.out.println("Data removed : " +
                        ↪  removeData);
                        break;
                    }

                case 3 :
                    Q.print();
                    break;

                case 4:
                    Q.peek();
                    break;

                case 5:
                    Q.clear();
                    break;
            }
        } while (choose <= 5 && choose >= 1);

        sc.close();
    }
```

```
    }
```

1.2.2 Result

Check if the result match with following image:

```
PS D:\Kuliah>  d:; cd 'd:\Kuliah'; & 'C:\Program
↪  Files\Java\jdk-18.0.2.1\bin\java.exe'
↪  '-XX:+ShowCodeDetailsInExceptionMessages' '-cp'
↪  'C:\Users\G4CE-PC\AppData\Roaming\Code\User\workspaceStorage\
↪  80d97a47d24665dc0bce7ab1e048ecbd\redhat.java\jdt_ws\Kuliah_28156aa7\bin'
↪  'QueueMain'
Insert maximum queue : 4
Choose menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
================================
1
Insert new data: 15
Choose menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
================================
1
Insert new data: 31
Choose menu:
1. Enqueue
2. Dequeue
3. Print
4. Peek
5. Clear
================================
4
The first element : 15
Choose menu:
1. Enqueue
2. Dequeue
```

```
3. Print
4. Peek
5. Clear
================================
```

1.2.3 Questions

1. In method create(), why is the front and rear attribute has initial value with 1 and not 0?
   Answer:
   The front and rear attribute is initialized with the number -1 to indicate that the queue is empty and there is no front index or rear index yet. This would make the array out of bound error when it apply to the array.

2. In method enqueue(), please explain the usage of this following code

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

   Answer:
   The if condition checks for the rear index. When it reach the maximum index of the queue, it will reset the rear index to 0. If the queue is full nothing would happend. But when the queue has more room, because the rear index has been reset to 0, the data will be insert at the rear which is at the first index.

3. Observe enqueue() method, which line of code indicates that the new data will be stored in last position of the queue?
   Answer:

   ```
   Q[rear] = data;
   ```

4. Observe dequeue() method, which line of code indicates that the data is removed in the first position of the queue?
   Answer:

   ```
   data = Q[front];
   ```

5. In dequeue method(), explain the usage of these codes !

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

`Answer:`
The if condition checks for the front index. When it reach the maximum index of the queue, it will reset the front index to 0. If the queue is full nothing would happend. But when the queue has more room, because the front index has been reset to 0, the data will be drop at the front which is at the first index.

6. In method print(), why the loop process has **int i = 0** instead of int **i=front**?
   `Answer:`
   Because the the first element of a queue doesn't necessarily start at the index 0. That is why queue keep the index of the first element instead of constantly rewriting the array as such that the first element is at the first index of the array.

7. In method print(), please explain why we insert this code in our program?

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

   `Answer:`
   The code is used to increment the i as such that the i will only cycle trough the index up to the maximum index and then return to 0 to then cycle again.

1.3 Lab Activities 2

In this lab activity, we will create a train ticket payment simple program with implementing the properties adjusted in railway stations environment

1.3.1 Steps

Passengers
name: String cityOrigin: String cityDestination: String ticketAmount: int price: int
Passengers(name: String, cityOrigin: String, cityDestination: String, ticketAmount: int, price: int)

1. Based on above class diagram, we will create a program written in Java
2. Create a new package named **Practicum2** and create a new class named **Passengers**
3. Add attributes for Passengers based on above class diagram, add the constructor as well

4. Copy the program code written in **Queue** in 1st Lab Activities to be reused in this package. We will need to modify the class, since the stored value in 1st lab activity is in integer data type, but we need it to store an object

5. Modify the class **Queue**, we change the data type **int[] Q** into **Passenger[] Q**. Since this case we will need to store Passenger object in the queue. In addition, we will need to modify **attributes, create(), enqueue(), and dequeue()**

```
int max, size, front, rear;
Passengers[] Q;

public void create() {
    Q = new Passengers[max];
    size = 0;
    front = rear = -1;
}

public void enqueue(Passengers data) {
    if (isFull()) {
        System.out.println("Queue is already full");
    } else {
        if (isEmpty()) {
            front = rear = 0;
        } else {
            if (rear == max - 1) {
                rear = 0;
            } else {
                rear++;
            }
        }
        Q[rear] = data;
        size++;
    }
}

public Passengers dequeue() {
    Passengers data = new Passengers("", "", "", 0, 0);
    if (isEmpty()) {
        System.out.println("Queue is still empty");
    } else {
        data = Q[front];
        size--;
        if (isEmpty()) {
```

```
            front = rear = -1;
        } else {
            if (front == max - 1) {
                front = 0;
            } else {
                front++;
            }
        }
    }
    return data;
}
```

6. Because one element in queue holds some information (name, cityOrigin, cityDestination, ticketAmount, price), we are needed to display all of that information. This leads to modifying the **peek() and print()** as well

```
public void peek() {
    if (!isEmpty()) {
        System.out.println("The first element : " + Q[front].name
        ↪  + " " + Q[front].cityOrigin + " " +
        ↪  Q[front].cityDestination + " " +
        ↪  Q[front].ticketAmount + " " + Q[front].price);
    } else {
        System.out.println("Queue is still empty");
    }
}

public void print() {
    if (isEmpty()) {
        System.out.println("Queue is still empty");
    } else {
        int i = front;
        while (i != rear) {
            System.out.println("The first element : " + Q[i].name
            ↪  + " " + Q[i].cityOrigin + " " +
            ↪  Q[i].cityDestination + " " + Q[i].ticketAmount +
            ↪  " " + Q[i].price);
            i = (i+1) % max;
        }
        System.out.println(Q[i].name + " " + Q[i].cityOrigin + "
        ↪  " + Q[i].cityDestination + " " + Q[i].ticketAmount +
        ↪  " " + Q[i].price);
        System.out.println("Element amount : " + size);
```

```
    }
}
```

7. Next, create a new class named **QueueMain** within the **Practicum2** package. Create **menu()** to provide menu options and allow the user to choose the menu when the program runs

```
public static void menu() {
    System.out.println("Choose menu: ");
    System.out.println("1. Queue");
    System.out.println("2. Dequeue");
    System.out.println("3. Check first queue");
    System.out.println("4. Check all queue");
    System.out.println("================================");
}
```

8. Create **main method** in the **QueueMain**, and declare the Scanner object with name **sc**

9. Create **max** variable to define the capacity of the queue. After that, instantiate queue object with name **queuePassenger** with its parameter is **max**

```
System.out.print("Insert maximum queue : ");
int max = sc.nextInt();
Queue queuePassenger = new Queue(max);
```

10. Declare a variable named **choose** with integer as its datatype to get which option did the user choose.

11. Add these following codes to loops menu options according to given input by the user.

```
int choose;
do {
    menu();
    choose = sc.nextInt();
    switch (choose) {
        case 1:
            System.out.print("Name: ");
            sc.nextLine();
            String name = sc.nextLine();
            System.out.print("City origin: ");
            String cityOrigin = sc.nextLine();
            System.out.print("City Destination: ");
            String cityDestination = sc.nextLine();
```

```
                System.out.print("Ticket Amount: ");
                int ticket = sc.nextInt();
                System.out.print("Price: ");
                int price = sc.nextInt();
                Passengers p = new Passengers(name, cityOrigin,
                ↪  cityDestination, ticket, price);
                sc.nextLine();
                queuePassenger.enqueue(p);
                break;

            case 2:
                Passengers data = queuePassenger.dequeue();
                if (!"".equals(data.name) &&
                ↪  !"".equals(data.cityOrigin) &&
                ↪  !"".equals(data.cityDestination) &&
                ↪  !"".equals(data.ticketAmount) &&
                ↪  !"".equals(data.price)) {
                    System.out.println("Data removed : " + data.name
                    ↪  + " " + data.cityOrigin + " " +
                    ↪  data.cityDestination + " " +
                    ↪  data.ticketAmount + " " + data.price);
                    break;
                }

            case 3:
                queuePassenger.peek();
                break;

            case 4:
                queuePassenger.print();
                break;

            case 5:
                queuePassenger.clear();
                break;
        }
    } while (choose <= 4 && choose >= 1);
```

12. Compile the program and run the **QueueMain** class. And observe the result

```
package practicum2;

public class Passengers {
```

```
    String name,
    cityOrigin,
    cityDestination;

    int ticketAmount,
    price;

    public Passengers(String name, String cityOrigin, String
    ↪  cityDestination, int ticketAmount, int price) {
        this.name = name;
        this.cityOrigin = cityOrigin;
        this.cityDestination = cityDestination;
        this.ticketAmount = ticketAmount;
        this.price = price;
    }
}
```

```
package practicum2;

public class Queue {
    int max, size, front, rear;
    Passengers[] Q;

    public Queue(int n) {
        max = n;
        create();
    }

    public void create() {
        Q = new Passengers[max];
        size = 0;
        front = rear = -1;
    }

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

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

public void peek() {
    if (!isEmpty()) {
        System.out.println("The first element : " +
        ↪  Q[front].name + " " + Q[front].cityOrigin + " " +
        ↪  Q[front].cityDestination + " " +
        ↪  Q[front].ticketAmount + " " + Q[front].price);
    } else {
        System.out.println("Queue is still empty");
    }
}

public void print() {
    if (isEmpty()) {
        System.out.println("Queue is still empty");
    } else {
        int i = front;
        while (i != rear) {
            System.out.println("The first element : " +
            ↪  Q[i].name + " " + Q[i].cityOrigin + " " +
            ↪  Q[i].cityDestination + " " +
            ↪  Q[i].ticketAmount + " " + Q[i].price);
            i = (i+1) % max;
        }
        System.out.println(Q[i].name + " " + Q[i].cityOrigin
        ↪  + " " + Q[i].cityDestination + " " +
        ↪  Q[i].ticketAmount + " " + Q[i].price);
        System.out.println("Element amount : " + size);
    }
}

public void clear() {
    if (!isEmpty()) {
        front = rear = -1;
        size = 0;
```

```
            System.out.println("Queue has been cleared
            ↪  successfully");
        } else {
            System.out.println("Queue is still empty");
        }
    }

    public void enqueue(Passengers data) {
        if (isFull()) {
            System.out.println("Queue is already full");
        } else {
            if (isEmpty()) {
                front = rear = 0;
            } else {
                if (rear == max - 1) {
                    rear = 0;
                } else {
                    rear++;
                }
            }
            Q[rear] = data;
            size++;
        }
    }

    public Passengers dequeue() {
        Passengers data = new Passengers("", "", "", 0, 0);
        if (isEmpty()) {
            System.out.println("Queue is still empty");
        } else {
            data = Q[front];
            size--;
            if (isEmpty()) {
                front = rear = -1;
            } else {
                if (front == max - 1) {
                    front = 0;
                } else {
                    front++;
                }
            }
        }
```

```
        return data;
    }
}

package practicum2;

import java.util.Scanner;

public class QueueMain {
    public static void menu() {
        System.out.println("Choose menu: ");
        System.out.println("1. Queue");
        System.out.println("2. Dequeue");
        System.out.println("3. Check first queue");
        System.out.println("4. Check all queue");
        System.out.println("================================");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Insert maximum queue : ");
        int max = sc.nextInt();
        Queue queuePassenger = new Queue(max);

        int choose;
        do {
            menu();
            choose = sc.nextInt();
            switch (choose) {
                case 1:
                    System.out.print("Name: ");
                    sc.nextLine();
                    String name = sc.nextLine();
                    System.out.print("City origin: ");
                    String cityOrigin = sc.nextLine();
                    System.out.print("City Destination: ");
                    String cityDestination = sc.nextLine();
                    System.out.print("Ticket Amount: ");
                    int ticket = sc.nextInt();
                    System.out.print("Price: ");
                    int price = sc.nextInt();
```

```
                    Passengers p = new Passengers(name,
                    ↪  cityOrigin, cityDestination, ticket,
                    ↪  price);
                    sc.nextLine();
                    queuePassenger.enqueue(p);
                    break;

                case 2:
                    Passengers data = queuePassenger.dequeue();
                    if (!"".equals(data.name) &&
                    ↪  !"".equals(data.cityOrigin) &&
                    ↪  !"".equals(data.cityDestination) &&
                    ↪  !"".equals(data.ticketAmount) &&
                    ↪  !"".equals(data.price)) {
                        System.out.println("Data removed : " +
                        ↪  data.name + " " + data.cityOrigin + "
                        ↪  " + data.cityDestination + " " +
                        ↪  data.ticketAmount + " " +
                        ↪  data.price);
                        break;
                    }

                case 3:
                    queuePassenger.peek();
                    break;

                case 4:
                    queuePassenger.print();
                    break;

                case 5:
                    queuePassenger.clear();
                    break;
            }
        } while (choose <= 4 && choose >= 1);

        sc.close();
    }
}
```

1.3.2 Result

Check if the result match with following image:

```
PS D:\Kuliah\Smt 2\Algoritma dan Struktur Data\Praktikum\Week 10\Queue>  d:; cd 'd:\Kuliah\Smt 2\Algoritma dan Struktur Data\Praktikum\Week 10\Queue'; & 'C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'D:\Kuliah\Smt 2\Algoritma dan Struktur Data\Praktikum\Week 10\Queue\bin' 'practicum2.QueueMain'
Insert maximum queue : 5
Choose menu:
1. Queue
2. Dequeue
3. Check first queue
4. Check all queue
================================
1
Name: Angga
City origin: Solo
City Destination: Sidoarjo
Ticket Amount: 2
Price: 176000
Choose menu:
1. Queue
2. Dequeue
3. Check first queue
4. Check all queue
================================
1
Name: Fadin
City origin: Banyuwangi
City Destination: Bandung
Ticket Amount: 1
Price: 65000
Choose menu:
1. Queue
2. Dequeue
3. Check first queue
4. Check all queue
================================
3
```

```
34 The first element : Angga Solo Sidoarjo 2 176000
```

1.3.3 Questions

1. In Queue Class, what's the function of this program code in method Dequeue?

   ```
   Passenger data = new Passenger("", "", "", 0, 0);
   ```

   Answer:
   It's used to insert a temporary empty object.

2. In previous number, if the program code changed to Passenger data = new Passenger() What will happen?
   Answer:
   Java return an error for an undefined constructor.

3. Show the program code used for displaying the data retrieved / removed from the queue!
   Answer:

   ```
   public Passengers dequeue() {
       Passengers data = new Passengers("", "", "", 0, 0);
       if (isEmpty()) {
           System.out.println("Queue is still empty");
       } else {
           data = Q[front];
           size--;
           if (isEmpty()) {
               front = rear = -1;
           } else {
               if (front == max - 1) {
                   front = 0;
               } else {
                   front++;
               }
           }
       }
       return data;
   }
   ```

4. Modify the program by adding a method named **peekRear()** in Queue class to check the last position within the queue. Add a menu for the user to perform and explore your program as well
   Answer:

```
public void peekRear() {
    if (!isEmpty()) {
        System.out.println("The last element : " + Q[rear].name +
        ↪  " " + Q[rear].cityOrigin + " " +
        ↪  Q[rear].cityDestination + " " + Q[rear].ticketAmount
        ↪  + " " + Q[rear].price);
    } else {
        System.out.println("Queue is still empty");
    }
}
```

5. Ensure that the **peekRear()** function can be executed inside the program Answer:

```
public static void menu() {
    System.out.println("Choose menu: ");
    System.out.println("1. Queue");
    System.out.println("2. Dequeue");
    System.out.println("3. Check first queue");
    System.out.println("4. Check last queue");
    System.out.println("5. Check all queue");
    System.out.println("================================");
}
```

```
switch (choose) {
    case 1:
        System.out.print("Name: ");
        sc.nextLine();
        String name = sc.nextLine();
        System.out.print("City origin: ");
        String cityOrigin = sc.nextLine();
        System.out.print("City Destination: ");
        String cityDestination = sc.nextLine();
        System.out.print("Ticket Amount: ");
        int ticket = sc.nextInt();
        System.out.print("Price: ");
        int price = sc.nextInt();
        Passengers p = new Passengers(name, cityOrigin,
        ↪  cityDestination, ticket, price);
        sc.nextLine();
        queuePassenger.enqueue(p);
        break;
```

```
            case 2:
                Passengers data = queuePassenger.dequeue();
                if (!"".equals(data.name) && !"".equals(data.cityOrigin)
                ↪  && !"".equals(data.cityDestination) &&
                ↪  !"".equals(data.ticketAmount) &&
                ↪  !"".equals(data.price)) {
                    System.out.println("Data removed : " + data.name + "
                    ↪  " + data.cityOrigin + " " + data.cityDestination
                    ↪  + " " + data.ticketAmount + " " + data.price);
                    break;
                }

            case 3:
                queuePassenger.peek();
                break;

            case 4:
                queuePassenger.peekRear();
                break;

            case 5:
                queuePassenger.print();
                break;

            case 6:
                queuePassenger.clear();
                break;
        }
```

1.4 Assignments

1. Add these 2 methods in **Queue** class in 1st practicum
2. Make a queue program for students when they need the signs for their KRS by the DPA. If the student is in queue, they will be required to fill in some information as follows:

Student
nim: String name: String classNumber: int gpa: double
Student (nim: String, name: String, classNumber: int,gpa: double)

Queue Class diagram:

Queue
max: int front: int rear: int size: int stdQueue: Student[]
Queue(max: int) create(): void isEmpty(): boolean isFull(): boolean enqueue(stdQueue: Student): void dequeue(): int print(): void peek(): void peekRear(): void peekPosition(nim: String): void printStudents(position: int): void

Notes:

- The implementation of Create(), isEmpty(), isFull(), enqueue(), dequeue() and print() functions are similar with we've built in practicum
- Peek() method is used for displaying students data in the first queue
- peekRead() method is used for displaying students data in the last queue
- peekPosition() method is used for displaying students data in the queue by their NIM
- printStudents() method is used for displaying a student data in specified position in a queue

```
package assignment;

public class Student {
```

```
    String name,
    nim;
    int classNumber;
    double gpa;

    public Student(String name, String nim, int classNumber, double
    ↪  gpa) {
        this.name = name;
        this.nim = nim;
        this.classNumber = classNumber;
        this.gpa = gpa;
    }
}
```

```
package assignment;
public class Queue {
    int max,
    front,
    rear,
    size;

    Student[] stdQueue;
    public Queue(int max) {
        this.max = max;
    }

    public void create() {
        stdQueue = new Student[max];
        size = 0;
        front = rear = -1;
    }

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

    public boolean isFull() {
        if (size == max) {
            return true;
```

```
        } else {
            return false;
        }
    }

    public void peek() {
        if (!isEmpty()) {
            System.out.println("The first element : " +
             ↪  stdQueue[front].name + " " + stdQueue[front].nim + " "
             ↪  + stdQueue[front].classNumber + " " +
             ↪  stdQueue[front].gpa);
        } else {
            System.out.println("Queue is still empty");
        }
    }

    public void peekRear() {
        if (!isEmpty()) {
            System.out.println("The last element : " +
             ↪  stdQueue[front].name + " " + stdQueue[front].nim + " "
             ↪  + stdQueue[front].classNumber + " " +
             ↪  stdQueue[front].gpa);
        } else {
            System.out.println("Queue is still empty");
        }
    }

    public void peekPosition(String nim) {
        boolean found = false;
        for (int i = front; i < max; i = (i + 1) % max) {
            if (stdQueue[i].nim == nim) {
                System.out.printf("student %s is in postion %d\n",
                 ↪  nim, i);
                found = true;
            }
        }
        if (!found) {
            System.out.println("Student not found in queue");
        }
    }

    public void print() {
```

```
        if (isEmpty()) {
            System.out.println("Queue is still empty");
        } else {
            int i = front;
            while (i != rear) {
                System.out.println("The first element : " +
                ↪  stdQueue[front].name + " " + stdQueue[front].nim +
                ↪  " " + stdQueue[front].classNumber + " " +
                ↪  stdQueue[front].gpa);
                i = (i+1) % max;
            }
            System.out.println(stdQueue[front].name + " " +
            ↪  stdQueue[front].nim + " " +
            ↪  stdQueue[front].classNumber + " " +
            ↪  stdQueue[front].gpa);
            System.out.println("Element amount : " + size);
        }
    }

    public void printStudents(int position) {
        for (int i = front; i < max; i = (i + 1) % max) {
            if (i == position) {
                System.out.println("student in position #" + position+
                ↪  " : " + stdQueue[front].name + " " +
                ↪  stdQueue[front].nim + " " +
                ↪  stdQueue[front].classNumber + " " +
                ↪  stdQueue[front].gpa);
            }
        }
    }

    public void enqueue(Student data) {
        if (isFull()) {
            System.out.println("Queue is already full");
        } else {
            if (isEmpty()) {
                front = rear = 0;
            } else {
                if (rear == max - 1) {
                    rear = 0;
                } else {
                    rear++;
```

```
                }
            }
            stdQueue[rear] = data;
            size++;
        }
    }

    public Student dequeue() {
        Student data = new Student("", "", 0, 0);
        if (isEmpty()) {
            System.out.println("Queue is still empty");
        } else {
            data = stdQueue[front];
            size--;
            if (isEmpty()) {
                front = rear = -1;
            } else {
                if (front == max - 1) {
                    front = 0;
                } else {
                    front++;
                }
            }
        }
        return data;
    }
}
```