Opgave 6: Queue

In deze opdrachten moesten we een queue maken bestaande uit een bepaald soort object. De queue had bepaalde methods waaronder de basismethods: elementen toevoegen achteraan de queue (enqueue) en vooraan een element van de queue afhalen (dequeue).

Interface

Het interface gedeelde van de opdracht hadden we gekregen en ziet er uit als volgt:

```
package com.company;

/** Queue ADT */
public interface IQueue<E> {
    /** Place an element at the rear of the queue.
     * @param it The element being enqueued. */
    public void enqueue(E it);
    /** Remove and return element at the front of the queue.
     @return The element at the front of the queue. */
    public E dequeue();
    /** @return The number of elements in the queue. */
    public int length();
}
```

De queue

We beginnen met het schrijven van de classe queue, hierbij schrijven we ook dat de interface van hiervoor wordt geïmplementeerd. We geven ook aan de queue mee over wat type elementen we er in gaan steken (E).

```
public class Queue<E> implements IQueue
```

Een paar integers die we gaan nodig hebben zijn front, rear, size en capacity. Tevens hebben we ook een array met elementen van het type E (hiervoor al gedefinieerd) .

```
public class Queue<E> implements IQueue
{

    private int front, rear, size=0, capacity;
    private E[] array;

    public Queue(int capacity) {
        this.capacity = capacity;
        front = this.size =0;
        rear = capacity-1;
        array = (E[])new Object[capacity];

    }
```

In de constuctor geven we de capaciteit van de rij mee, dit zegt hoeveel elementen er maximaal in de queue mogen worden opgeslagen. We geven ook de plaats van het begin van de rij aan met front (is nul) en ook het einde can de rij, rear. Dan maken we een array die capaciteit groot is en gevuld zal worden met elementen van het type E.

```
boolean isEmpty(Queue tempQueue)
{
    return(tempQueue.size==0);
}

private boolean isFull(Queue tempQueue)
{
    return (tempQueue.size == tempQueue.capacity);
}
```

De twee methods hierboven geven meer informatie over de queue, isEmpty zal een boolean teruggeven waarin er wordt gezegd of de rij leeg is. Dit doen we door te kijken of de size (het aantal element in de array) gelijk is aan nul.

De tweede method die we toevoegen is het tegenovergestelde, we kijken of de rij vol zit door te kijken of het aantal elementen die er nu in zitten evenveel is als het maximum elementen die er in mogen zitten (capacity).

Enqueue

Hier gaan we elementen achteraan de rij toevoegen.

```
@Override
public void enqueue(Object it)
{
    if (isFull( tempQueue: this))
        System.out.println("Queue is full, item has been put in front");

    this.rear = (this.rear + 1)%this.capacity;
    this.array[this.rear] = (E) it;
    this.size++;
}
```

We gaan eerst kijken of de rij niet vol zit, is dit wel het geval dan printen we een berichtje af. Is dit niet het geval dan wordt het nieuwe einde van de rij het vorige plus 1 met daarvan de modulo van de capaciteit. Dit doen we zodat als de rij vol zit het nieuwe element vooraan wordt gezet. Daarna voegen we het nieuwe element it in in de array op de plaats van het nieuwe einde. We geven ook nog mee dat de grootte van de rij met 1 wordt vergroot.

Dequeu

Hier halen we elementen vooraan de rij er terug af. We krijgen dan ook het object zijn informatie te zien dat vooraan stond.

```
@Override
public Object dequeue()
{
    if(isEmpty( tempQueue: this))
    {
        System.out.println("Queue is empty.");
        return null;
    }
    else
    {
        Object obj = this.array[this.front];
        this.front = (this.front +1 )%this.capacity;
        this.size --;
        return obj;
    }
}
```

Hier gaan we eerst kijken of de queue niet al leeg is, we sturen een bericht als dit het geval zou zijn en dan stopt de method, we geven niets terug naar buiten. Is dit niet dan wordt het object dat we teruggeven degene die op plaats front staat van de array. Daarna verklaren we weer wat de nieuwe front wordt en maken we de size kleiner met 1. We geven dan ook het object terug met return.

Length

Een derde method is length, deze zal een integer teruggeven die ons verteld hoe groot de queue is.

```
@Override
public int length()
{
    int len = this.size;

    return len;
}
```

Testen

Voor het nakijken van de queue heb ik zoals w vorig jaar deden een testclasse gemaakt die nagaat of we de juiste waarden terugkrijgen.

```java
package com.company;

import org.junit.Test;
import static org.junit.Assert.*;

public class TestQueue {

    @Test
    public void test_enqueue ()
    {
        Queue<Integer> testQ= new Queue<>( capacity: 10);
        assertEquals(testQ.length(), actual: 0);

        testQ.enqueue( it: 6);
        assertEquals(testQ.length(), actual: 1);
        testQ.enqueue( it: 12);
        assertEquals(testQ.length(), actual: 2);
    }
```

We testen hier eerst de methode enqueue. We maken een queue met elementen van het type Integer en geven het een capaciteit van 10.

We gebruiken de method length en we willen 0 terugkrijgen omdat de rij nog leeg is. We steken de waarde 6 toe aan de rij met enqueue en we willen nu dat de lengte van de rij gelijk is aan 1. Dit doen we nog eens met waarde 12.

```java
@Test
public void test_dequeue()
{
    Queue<Integer> testQ= new Queue<>( capacity: 10);
    testQ.enqueue( it: 0);
    testQ.enqueue( it: 1);
    testQ.enqueue( it: 2);

    assertEquals(testQ.length(), actual: 3);

    testQ.dequeue();
    assertEquals(testQ.dequeue(), actual: 1);
    assertEquals(testQ.dequeue(), actual: 2);
    assertNull(testQ.dequeue());

}
```

Een tweede test dat we doen is voor de dequeue method. We maken eerst een rij aan en vullen respectievelijk met de waarden 0,1,2. We kijken dan of de lengte inderdaad gelijk is aan drie. We gaan eerst één keer dequeuen waarbij de waarde 1 nu vooraan zou moeten staan. We testen dit met assertEquals of

we bij de volgende dequeue 1 terugkrijgen. Op het einde gaan we ook na of we niets terugkrijgen als we de method dequeue gebruiken wanneer de rij leeg is.

```
package com.company;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class Main {

    public static void main(String[] args) {

        Result result = JUnitCore.runClasses(TestQueue.class);

        for(Failure fail : result.getFailures())
        {
            System.out.println(fail.toString());
        }
        if (result.wasSuccessful())
            System.out.println("Everything went successfully.");
        else
            System.out.println("Not all test were correct.");
    }
}
```

In de main slaan we de resultaten van de testklasse op in result. Met een for each lus gaan we over alle falen die we uit de resultaten halen en laten deze afprinten.

Wanneer alles correct is verlopen laten we dat even weten met een message. Bij het runnen van het programma krijgen we volgende als output.

```
Queue is empty.
Everything went successfully.
```

De eerste zin komt van wanneer we op een lege rij de method dequeue toepasten. De tweede zin verteld ons dat alle test succesvol zijn verlopen en dat de queue werkt zoals gehoopt!