# Task 2.1

## Source Code

public class Stack {

static int *stackSize* = 5; //The number of integers that can be stored in a stack at maximum

static int *topOfStack* = -1; //The index of the last element added to the stack

static int[] *stack* = new int[*stackSize*]; //The fixed array that holds the elements of the stack

static boolean *errorFree* = true; //Boolean to deduce whether the stack hasn't encountered an error

public static boolean isEmpty () {

return *topOfStack* == -1;

}

public static boolean isFull () {

return *topOfStack* == *stackSize* - 1;

}

public static void empty () {

*errorFree* = true;

*topOfStack* = -1;

}

public static int top () {

// Return the integer held at the top of stack if the stack is error free and not empty

*errorFree* = ! (*isEmpty* ()) & *errorFree*;

if (*errorFree*) {

return *stack*[*topOfStack*]; //Returns integer at the index of the fixed array indicated by topOfStack

} else {

return 0; //Error case – returns 0

}

}

public static void push (int value) {

/\* Adds a number to the stack and sets the top of stack to the index of the number you have added if the stack

\* is not full and there are no errors

\*/

*errorFree* = ! (*isFull* ()) & *errorFree*;

if (*errorFree*) { //If error free, does as indicated, if not doesn't do anything

*topOfStack* = *topOfStack* + 1;

*stack*[*topOfStack*] = value;

}

}

public static void pop () {

/\* Deletes the element at the top of the stack and deducts one from the stack index if the stack is not empty

\* and there are no errors

\*/

*errorFree* = ! (*isEmpty* ()) & *errorFree*;

if (*errorFree*) {

*topOfStack* = *topOfStack* - 1;

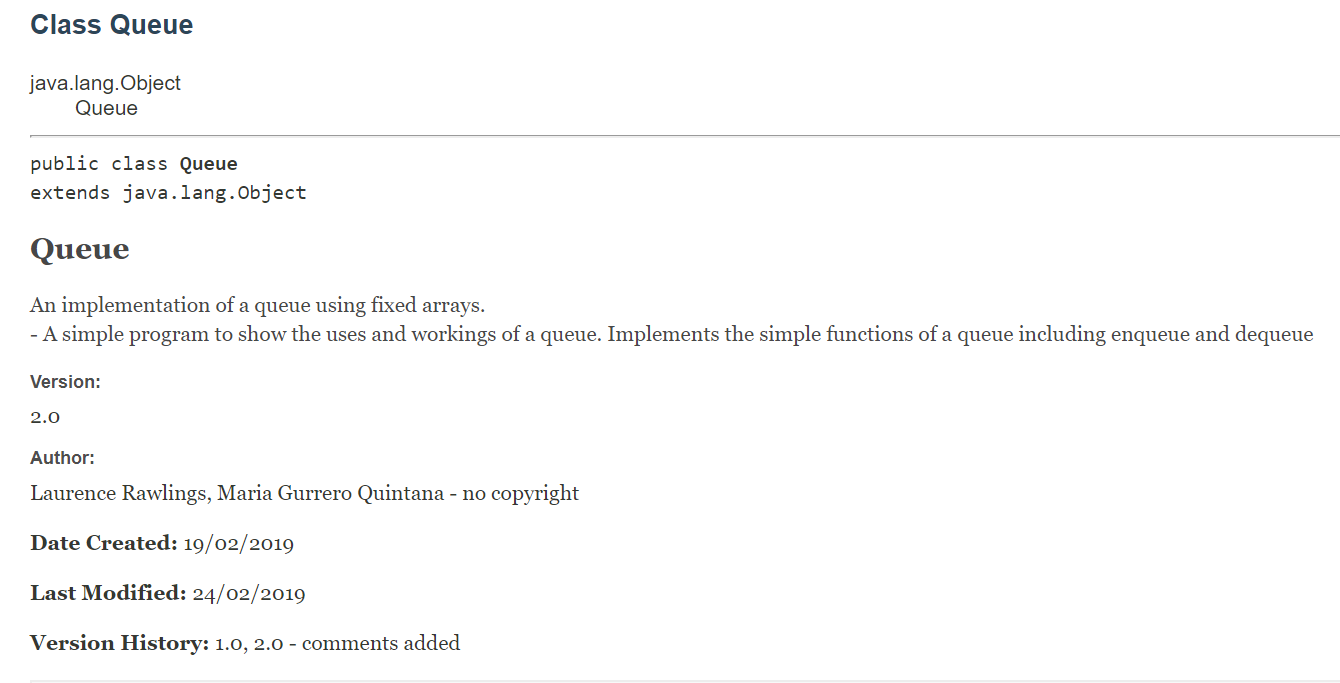
}

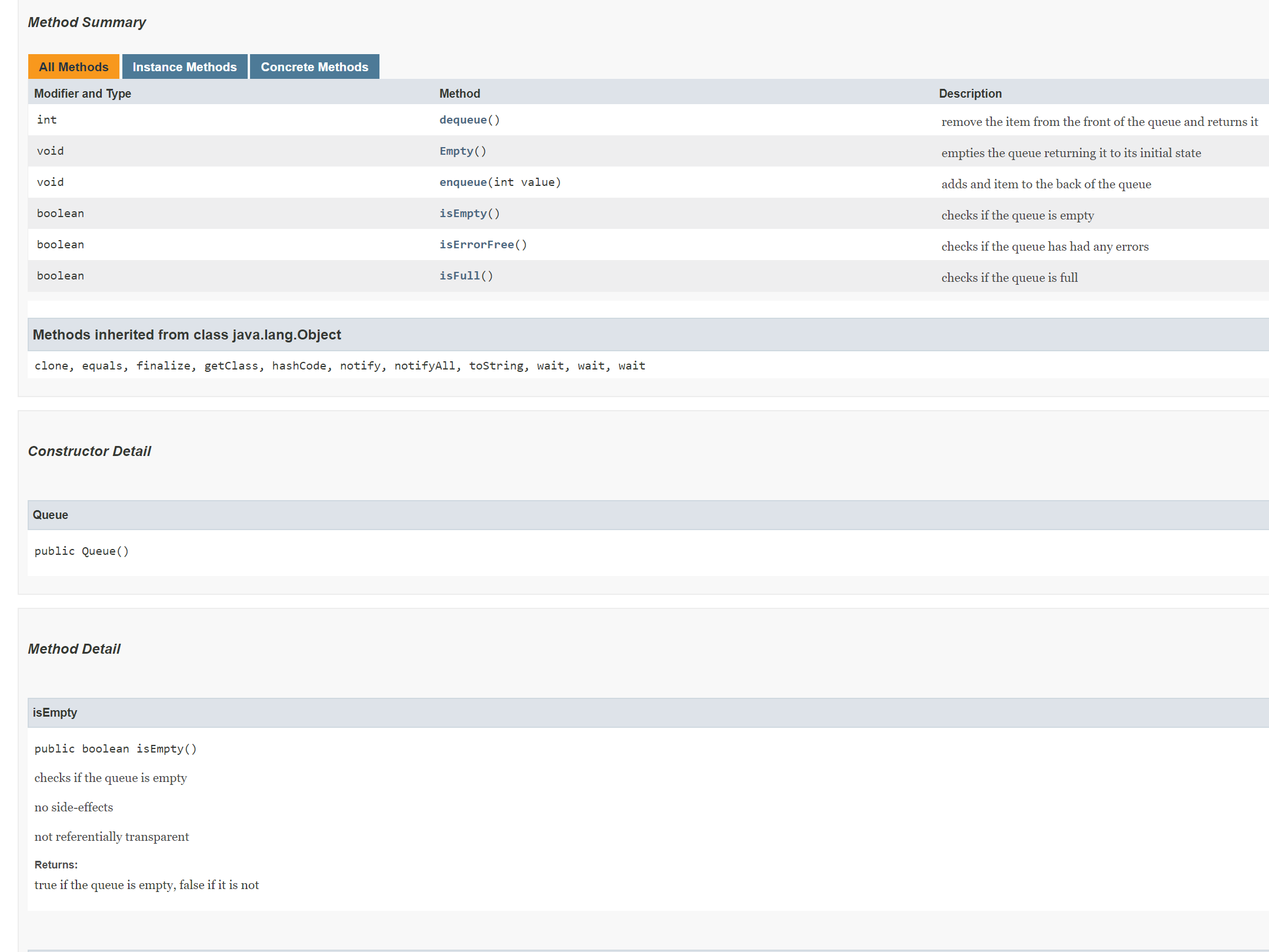
}

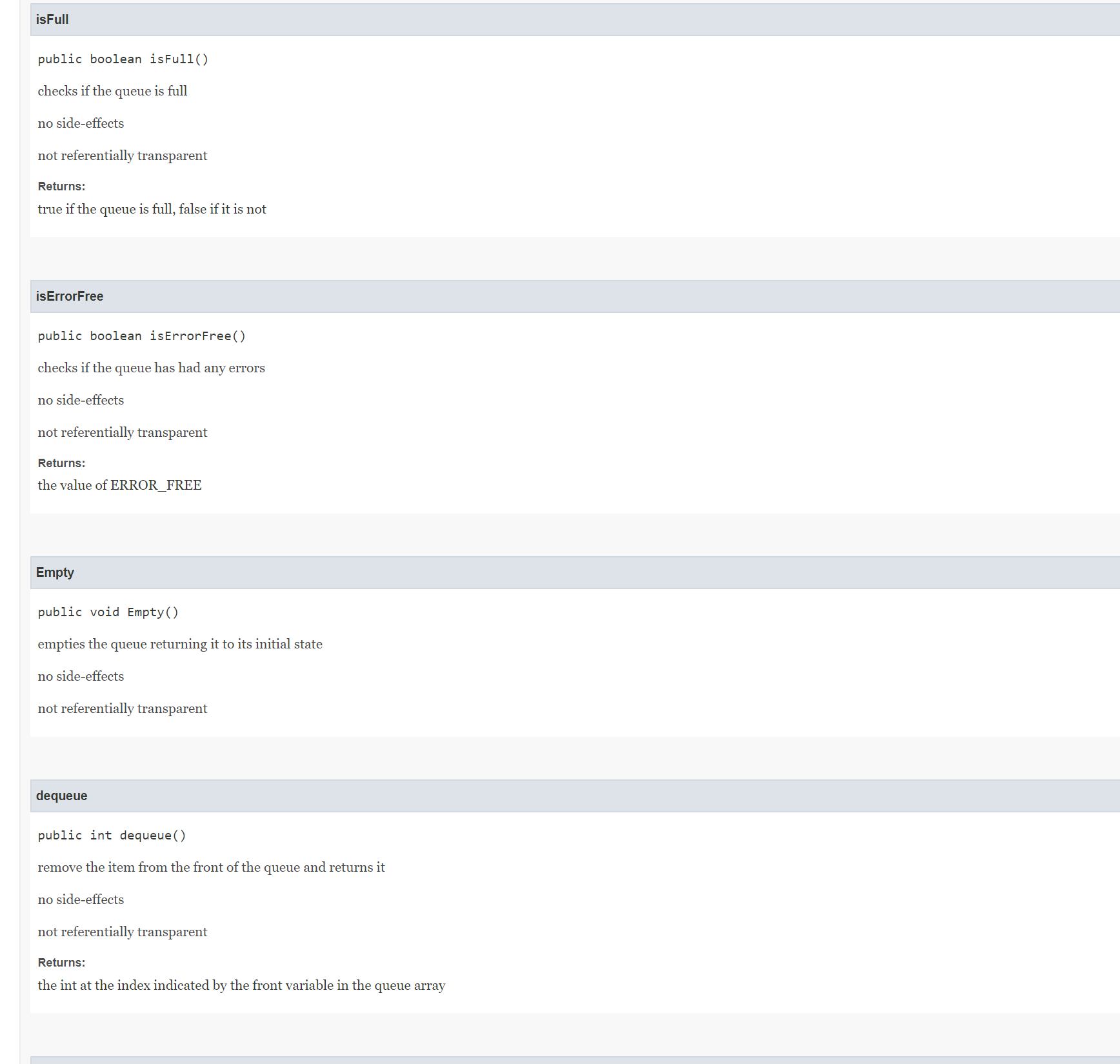
}

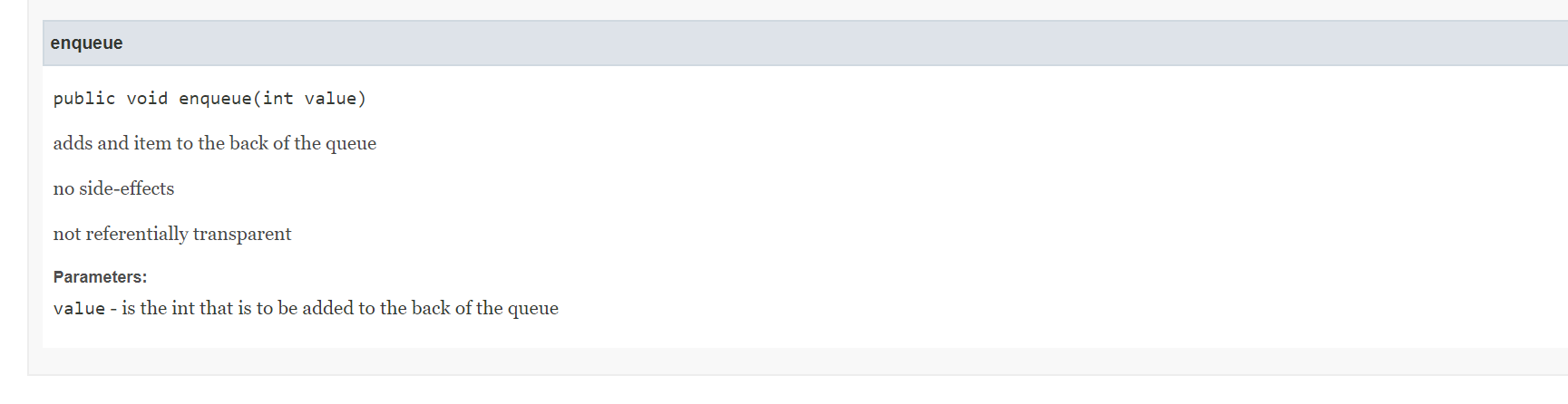
# Task 2.2

## Javadoc









## Source Code

*/\*\*  
 \* <h1>Queue</h1>  
 \* <p>An implementation of a queue using fixed arrays.<br>  
 \* - A simple program to show the uses and workings of a queue. Implements*

*\* the simple functions of a queue including enqueue and dequeue</p>  
 \*  
 \** ***@author*** *Laurence Rawlings, Maria Gurrero Quintana  
 \* - no copyright  
 \*  
 \* <p><b>Date Created: </b> 19/02/2019</p>  
 \* <p><b>Last Modified: </b> 24/02/2019</p>  
 \* <p><b>Version History: </b>1.0, 2.0 - comments added</p>  
 \*  
 \** ***@version*** *2.0  
 \*/*public class Queue {  
 private int front = *queueSize* - 1, back = *queueSize* - 1, length = 0;  
  
 private int[] queue = new int[*queueSize*]; //A fixed array queue of size queueSize, in this case 5  
 private boolean ERROR\_FREE = true; //Boolean to deduce whether the array is error free  
  
 private final static int *queueSize* = 5; //Private int stating the size of the array queue  
  
 */\*\*  
 \* checks if the queue is empty  
 \* <p> no side-effects </p>  
 \* <p> not referentially transparent </p>  
 \** ***@return*** *true if the queue is empty, false if it is not  
 \*/* public boolean isEmpty() {  
 return length == 0;  
 }  
  
 */\*\*  
 \* checks if the queue is full  
 \* <p> no side-effects </p>  
 \* <p> not referentially transparent </p>  
 \** ***@return*** *true if the queue is full, false if it is not  
 \*/* public boolean isFull() {  
 return (length == *queueSize*);  
 }  
  
 */\*\*  
 \* checks if the queue has had any errors  
 \* <p> no side-effects </p>  
 \* <p> not referentially transparent </p>  
 \** ***@return*** *the value of ERROR\_FREE  
 \*/* public boolean isErrorFree() {  
 return ERROR\_FREE;  
 }  
  
 */\*\*  
 \* empties the queue returning it to its initial state  
 \* <p> no side-effects </p>  
 \* <p> not referentially transparent </p>  
 \*/* public void Empty() {  
 front = *queueSize* - 1;  
 back = *queueSize* - 1;  
 length = 0;  
 ERROR\_FREE = true;  
 }  
  
 /\* If the array is error free, reduce the array length by one.  
 \* Remove the integer at the front of the array  
 \*/  
  
  
 */\*\*  
 \* remove the item from the front of the queue and returns it  
 \* <p> no side-effects </p>  
 \* <p> not referentially transparent </p>  
 \** ***@return*** *the int at the index indicated by the front variable in the*

*\* queue array  
 \*/* public int dequeue() {  
 ERROR\_FREE = !(isEmpty()) & (ERROR\_FREE);  
 if (ERROR\_FREE) {  
 length--;  
  
 /\* If the front of the queue is at the last position of the array

\* and an item is dequeued, move the front position of the queue to

\* the start of the array  
 \*/  
  
 if (front == *queueSize* - 1) {  
 front = 0;  
 } else {  
 front++;  
 }  
 return queue[front];  
 } else {  
 return 0;  
 }  
 }  
  
 */\*\*  
 \* adds and item to the back of the queue  
 \* <p> no side-effects </p>  
 \* <p> not referentially transparent </p>  
 \** ***@param*** *value is the int that is to be added to the back of the queue  
 \*/* public void enqueue(int value) {  
 ERROR\_FREE = !(isFull()) & ERROR\_FREE;  
 if (ERROR\_FREE) {  
 length++;  
  
 /\* If the back of the queue is at the last position of the array

\* and an item is dequeued, move the back position of the queue to

\* the front of the array  
 \*/  
  
 if (back == *queueSize* - 1) {  
 back = 0;  
 } else {  
 back++;  
 }  
 queue[back] = value;  
 }  
 }  
}