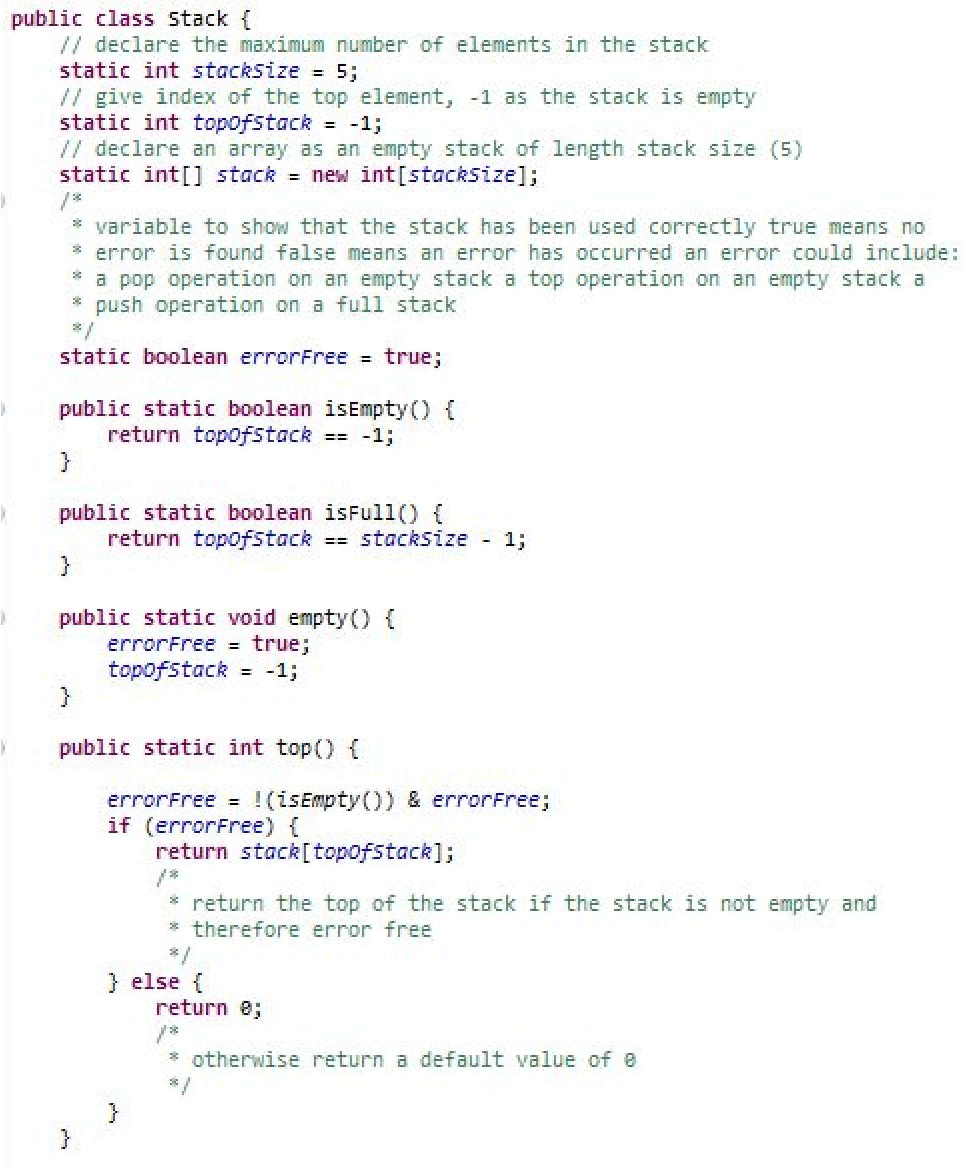
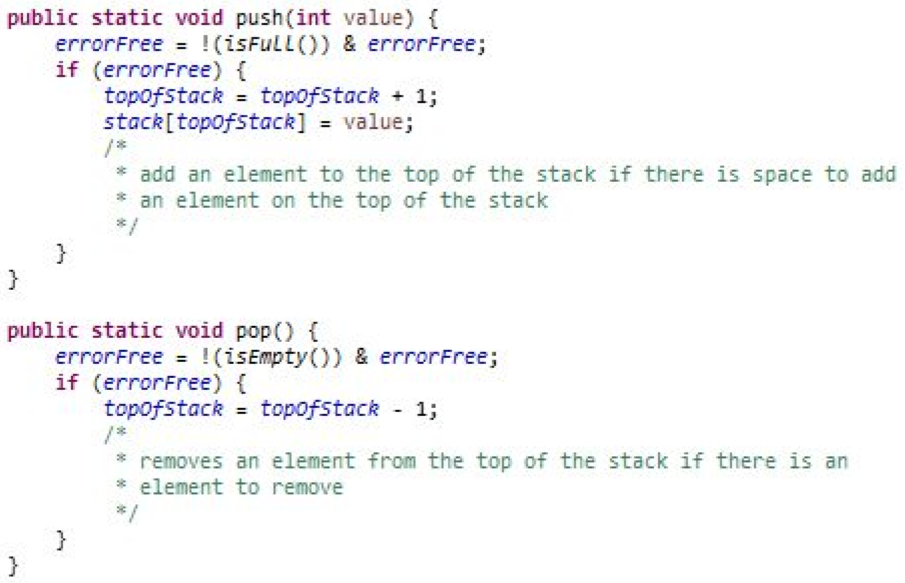
Task 2.1

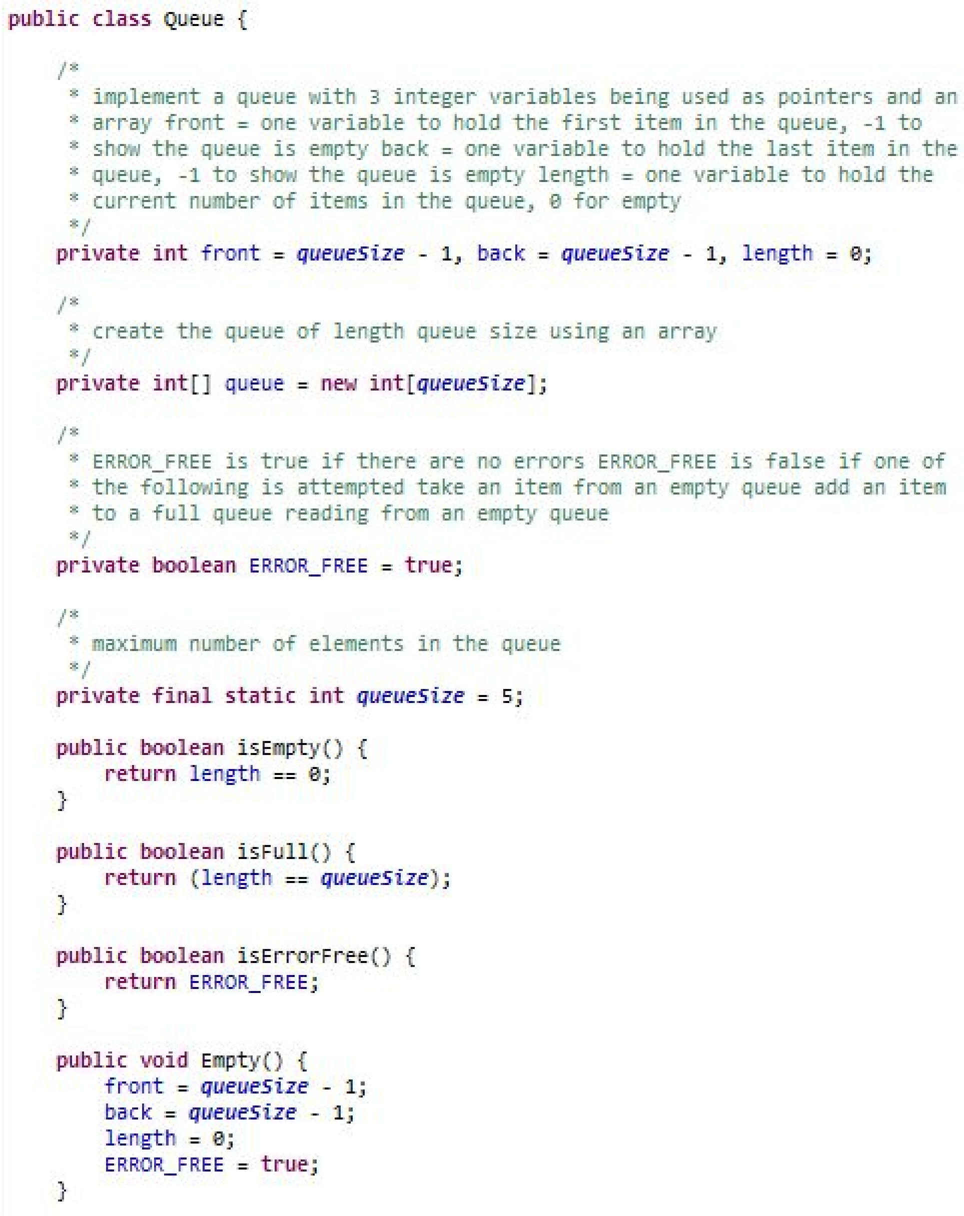


```
public class Stack {
    // declare the maximum number of elements in the stack
    static int stackSize = 5;
    // give index of the top element, -1 as the stack is empty
    static int topOfStack = -1;
    // declare an array as an empty stack of length stack size (5)
    static int[] stack = new int[stackSize];
    /*
     * variable to show that the stack has been used correctly true means no
     * error is found false means an error has occurred an error could include:
     * a pop operation on an empty stack a top operation on an empty stack a
     * push operation on a full stack
     */
    static boolean errorFree = true;

    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() {

        errorFree = !(isEmpty()) & errorFree;
        if (errorFree) {
            return stack[topOfStack];
            /*
             * return the top of the stack if the stack is not empty and
             * therefore error free
             */
        } else {
            return 0;
            /*
             * otherwise return a default value of 0
             */
        }
    }
```

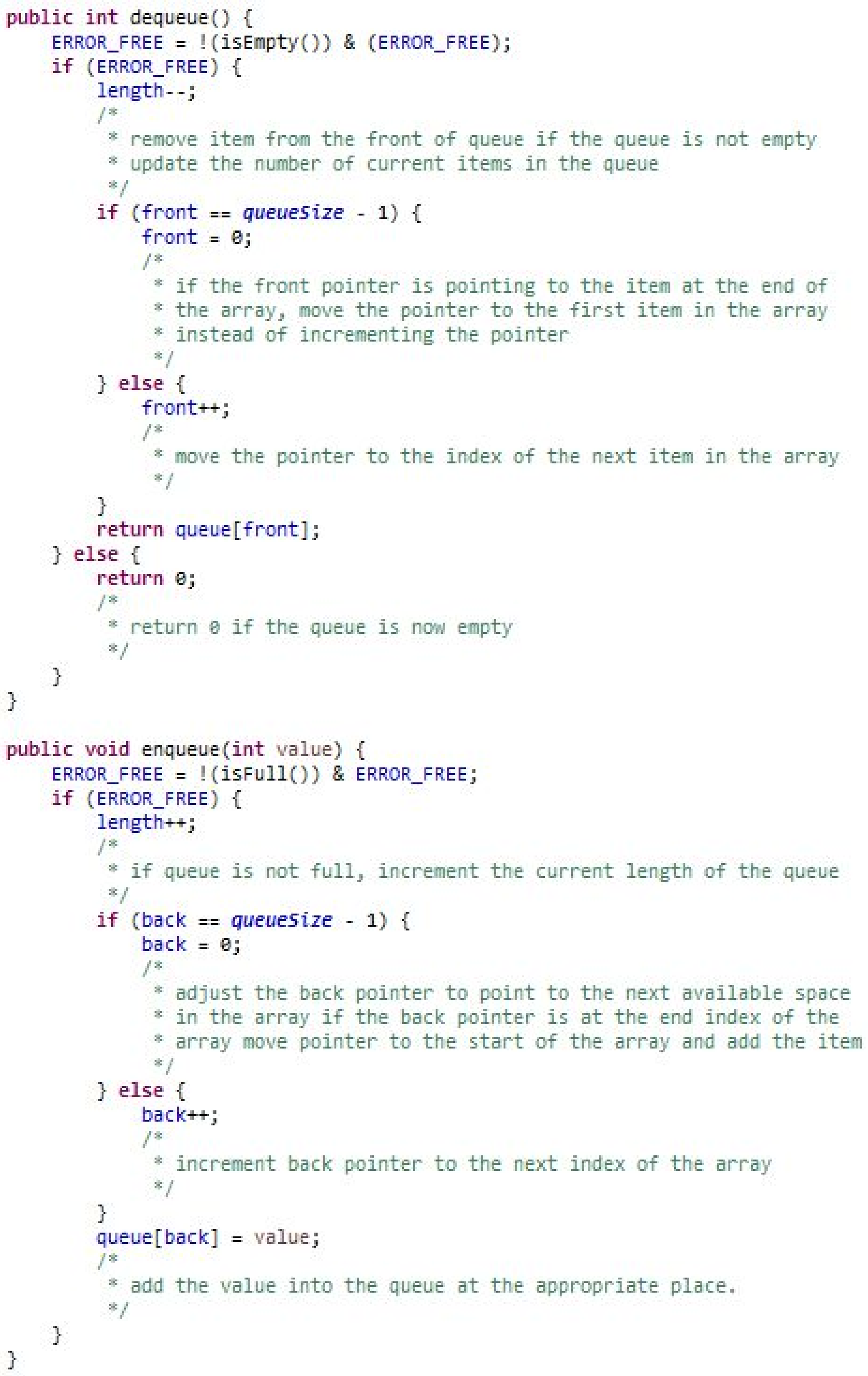


```
public static void push(int value) {
    errorFree = !(isFull()) & errorFree;
    if (errorFree) {
        topOfStack = topOfStack + 1;
        stack[topOfStack] = value;
        /*
         * add an element to the top of the stack if there is space to add
         * an element on the top of the stack
         */
    }
}

public static void pop() {
    errorFree = !(isEmpty()) & errorFree;
    if (errorFree) {
        topOfStack = topOfStack - 1;
        /*
         * removes an element from the top of the stack if there is an
         * element to remove
         */
    }
}
```

Task 2.2



```
public class Queue {

    /*
     * implement a queue with 3 integer variables being used as pointers and an
     * array front = one variable to hold the first item in the queue, -1 to
     * show the queue is empty back = one variable to hold the last item in the
     * queue, -1 to show the queue is empty length = one variable to hold the
     * current number of items in the queue, 0 for empty
     */
    private int front = queueSize - 1, back = queueSize - 1, length = 0;

    /*
     * create the queue of length queue size using an array
     */
    private int[] queue = new int[queueSize];

    /*
     * ERROR_FREE is true if there are no errors ERROR_FREE is false if one of
     * the following is attempted take an item from an empty queue add an item
     * to a full queue reading from an empty queue
     */
    private boolean ERROR_FREE = true;

    /*
     * maximum number of elements in the queue
     */
    private final static int queueSize = 5;

    public boolean isEmpty() {
        return length == 0;
    }

    public boolean isFull() {
        return (length == queueSize);
    }

    public boolean isErrorFree() {
        return ERROR_FREE;
    }

    public void Empty() {
        front = queueSize - 1;
        back = queueSize - 1;
        length = 0;
        ERROR_FREE = true;
    }
```



```
public int dequeue() {
    ERROR_FREE = !(isEmpty()) & (ERROR_FREE);
    if (ERROR_FREE) {
        length--;
        /*
         * remove item from the front of queue if the queue is not empty
         * update the number of current items in the queue
         */
        if (front == queueSize - 1) {
            front = 0;
            /*
             * if the front pointer is pointing to the item at the end of
             * the array, move the pointer to the first item in the array
             * instead of incrementing the pointer
             */
        } else {
            front++;
            /*
             * move the pointer to the index of the next item in the array
             */
        }
        return queue[front];
    } else {
        return 0;
        /*
         * return 0 if the queue is now empty
         */
    }
}

public void enqueue(int value) {
    ERROR_FREE = !(isFull()) & ERROR_FREE;
    if (ERROR_FREE) {
        length++;
        /*
         * if queue is not full, increment the current length of the queue
         */
        if (back == queueSize - 1) {
            back = 0;
            /*
             * adjust the back pointer to point to the next available space
             * in the array if the back pointer is at the end index of the
             * array move pointer to the start of the array and add the item
             */
        } else {
            back++;
            /*
             * increment back pointer to the next index of the array
             */
        }
        queue[back] = value;
        /*
         * add the value into the queue at the appropriate place.
         */
    }
}
```