

## ASSIGNMENT 2 FRONT SHEET

<b>Qualification</b>	<b>BTEC Level 5 HND Diploma in Computing</b>		
<b>Unit number and title</b>	Unit 19: Data Structures and Algorithms		
<b>Submission date</b>	August 19 <sup>th</sup> , 2021	<b>Date Received 1st submission</b>	August 19 <sup>th</sup> , 2021
<b>Re-submission Date</b>	August 25 <sup>th</sup> , 2021	<b>Date Received 2nd submission</b>	August 21 <sup>st</sup> , 2021
<b>Student Name</b>	Pham Manh Quan	<b>Student ID</b>	BHAF190226
<b>Class</b>	BH-AF-2005-2.3	<b>Assessor name</b>	Ngo Thi Mai Loan
<b>Student declaration</b>  I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I understand that making a false declaration is a form of malpractice.			
		<b>Student's signature</b>	

### Grading grid

P4	P5	P6	P7

☐ **Summative Feedback:**

☐ **Resubmission Feedback:**

**Grade:**

**Assessor Signature:**

**Date:**

**Internal Verifier's Comments:**

**IV Signature:**

## Contents

<b>INTRODUCE</b> .....	5
<b>CONTENT</b> .....	5
<b>I. Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem.</b> .....	5
1.1 ADT design.....	5
1.2. Flowchart .....	9
1.3. Execute ADT .....	10
<b>II. Perform error handling and test results</b> .....	15
2.1. Java try and catch.....	15
2.2. try ... catch when execute.....	19
2.3. Test case.....	20
<b>III. The asymptotic analysis technique is used to evaluate the efficiency of the algorithm</b> .....	21
3.1 What is Algorithmic Analysis?.....	21
3.2 What is asymptotic analysis?.....	22
3.3 Big Oh Notation, in Data Structures and Algorithms.....	24
3.4 Omega Notation, in Data Structures and Algorithms.....	25
3.5 Theta Notation, in Data Structures and Algorithms .....	26
<b>IV. Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.</b> .....	26
4.1. What is space complexity? .....	26
4.2. What is time complexity?.....	28
<b>CONCLUDED</b> .....	31
<b>REFERENCES</b> .....	31

## List of Figure

Figure 1 ADT design.....	5
Figure 2 Class Node .....	6
Figure 3 Class MyQueue .....	6
Figure 4 Class MyStack .....	7
Figure 5 Main program.....	8

Figure 6 option 1 .....	9
Figure 7 option 2 .....	9
Figure 8 option 3 .....	10
Figure 9 Execute ADT .....	10
Figure 10 Press 1 to enter message .....	11
Figure 11 Press 2 to send the message .....	11
Figure 12 The message has been processed and sent .....	12
Figure 13 Press 3 to receive the message .....	13
Figure 14 the received message .....	14
Figure 15 Java try and catch .....	15
Figure 16 class MyQueue .....	16
Figure 17 Class MyStack .....	17
Figure 18 Class Main .....	18
Figure 19 Before use try ... catch .....	19
Figure 20 After use try ... catch .....	19
Figure 21 Algorithmic Analysis .....	21
Figure 22 asymptotic analysis .....	22
Figure 23 Big O Notation .....	23
Figure 24 Notation $\Omega$ .....	24
Figure 25 Theta Notation .....	26
Figure 26 space complexity .....	27
Figure 27 Example with simple algorithm .....	27
Figure 28 Example with for loop .....	28
Figure 29 time complexity .....	29
Figure 30 Example with simple algorithm .....	29
Figure 31 Example with for loop .....	30
Figure 32 Example with nested loop .....	30

## INTRODUCE

In this assignment, I will implement a complex ADT and algorithm in a programming language (Java) to solve a well-defined problem then I will implement error handling and report the test results and finally, I will give two possible ways of measuring the effectiveness of an algorithm with specific examples.

## CONTENT

### I. Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem.

#### 1.1 ADT design

In the article, I discussed the following options:

- Press 1 to add a new message. First, we add the message, type “exit” to
- end, if the message exceeds 250 characters, the message...
- Press 2 to send
- Press 3 to display received messages
- Press 4 to exit

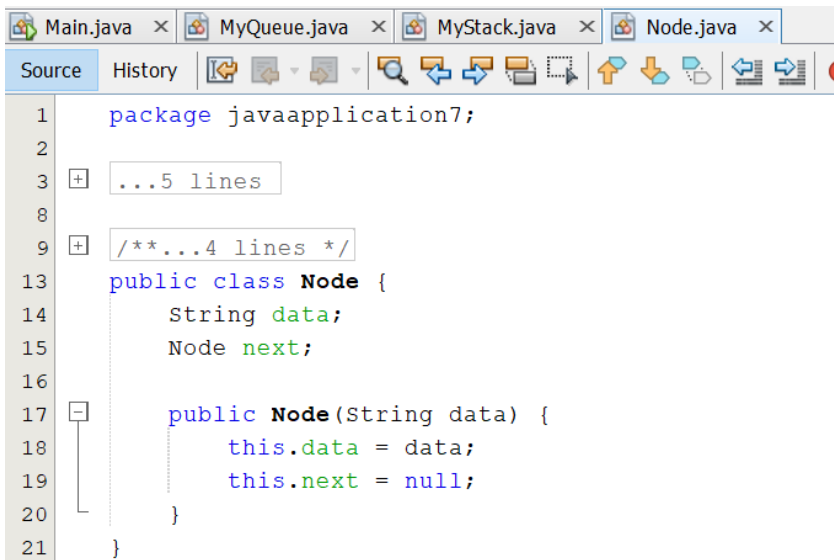
```
* @param args the command line arguments
*/
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    MyStack ms = new MyStack();
    MyQueue mq = new MyQueue();

    try {
        OUTER:
        while (true) {
            System.out.println("Xin mời chọn");
            System.out.println("1. Nhập tin nhắn");
            System.out.println("2. Gửi tin nhắn");
            System.out.println("3. Hiện thị tin nhắn nhận được");
            System.out.println("4. Thoát");
```

*Figure 1 ADT design*

I will use Linked list to design for Node class and MyQueue class. So all messages will be stored in class MyQueue.

## - Class Node



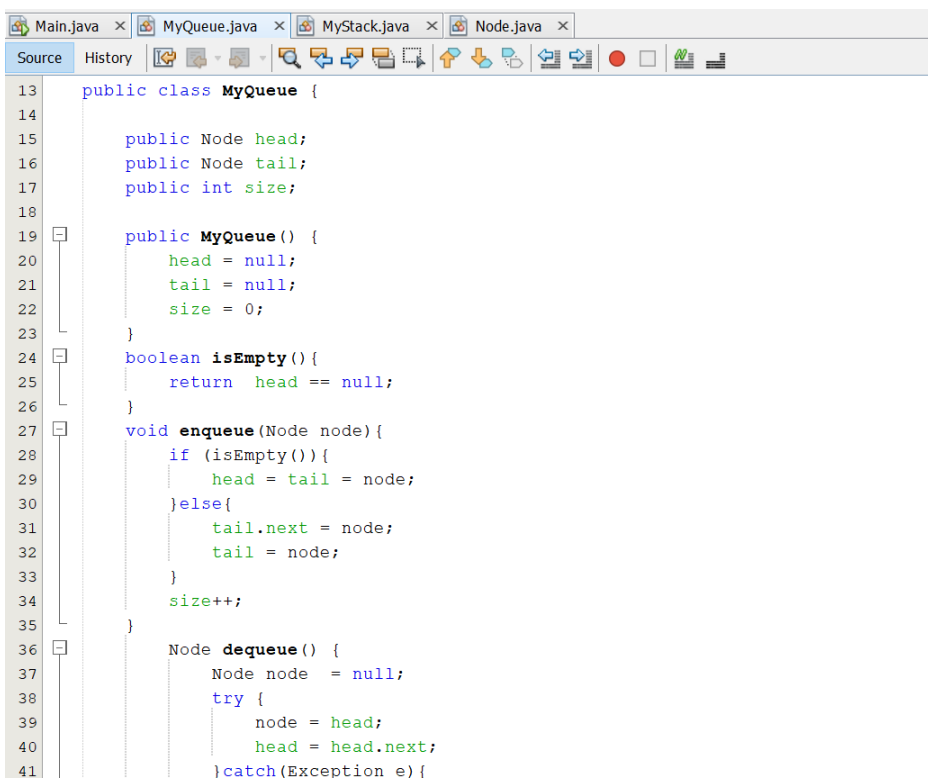
```

1  package javaapplication7;
2
3  ... 5 lines
8
9  /**... 4 lines */
13 public class Node {
14     String data;
15     Node next;
16
17     public Node(String data) {
18         this.data = data;
19         this.next = null;
20     }
21 }

```

Figure 2 Class Node

## - Class MyQueue



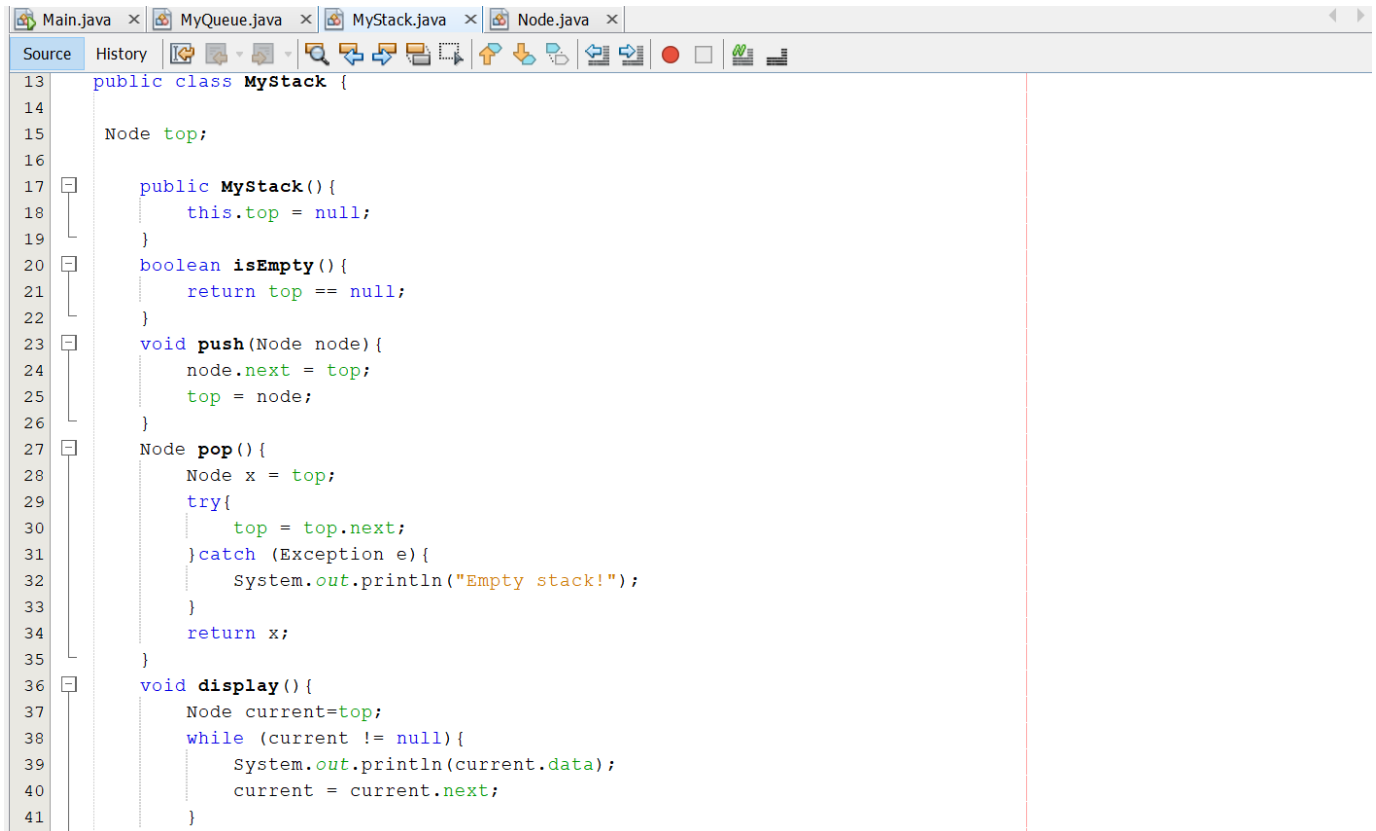
```

13 public class MyQueue {
14
15     public Node head;
16     public Node tail;
17     public int size;
18
19     public MyQueue() {
20         head = null;
21         tail = null;
22         size = 0;
23     }
24     boolean isEmpty() {
25         return head == null;
26     }
27     void enqueue(Node node) {
28         if (isEmpty()) {
29             head = tail = node;
30         } else {
31             tail.next = node;
32             tail = node;
33         }
34         size++;
35     }
36     Node dequeue() {
37         Node node = null;
38         try {
39             node = head;
40             head = head.next;
41         } catch (Exception e) {

```

Figure 3 Class MyQueue

## - Class MyStack



```

13 public class MyStack {
14
15     Node top;
16
17     public MyStack() {
18         this.top = null;
19     }
20     boolean isEmpty() {
21         return top == null;
22     }
23     void push(Node node) {
24         node.next = top;
25         top = node;
26     }
27     Node pop() {
28         Node x = top;
29         try {
30             top = top.next;
31         } catch (Exception e) {
32             System.out.println("Empty stack!");
33         }
34         return x;
35     }
36     void display() {
37         Node current = top;
38         while (current != null) {
39             System.out.println(current.data);
40             current = current.next;
41         }

```

Figure 4 Class MyStack

And finally, the code of the main program

```

Main.java x MyQueue.java x MyStack.java x Node.java x
Source History
18
19 public static void main(String[] args) {
20     Scanner sc = new Scanner(System.in);
21     MyStack ms = new MyStack();
22     MyQueue mq = new MyQueue();
23
24     try {
25         OUTER:
26         while (true) {
27             System.out.println("Xin mời chọn");
28             System.out.println("1. Nhập tin nhắn");
29             System.out.println("2. Gửi tin nhắn");
30             System.out.println("3. Hiển thị tin nhắn nhận được");
31             System.out.println("4. Thoát");
32
33             System.out.print("Lựa chọn của bạn : ");
34             int choice=sc.nextInt();
35             switch (choice) {
36                 case 1 -> {
37                     System.out.println("Mời bạn nhập tin, Gõ 'exit' để thoát");
38                     String mess="";
39                     while (!mess.equals("exit")){
40                         mess=sc.nextLine();
41                         if(!mess.equals("exit") && !mess.isEmpty() && mess.length() < 250){
42                             mq.enqueue(new Node(mess));
43                         }
44                     }
45                 }
46                 case 2 -> {
47                     while (!mq.isEmpty()){
48                         ms.push(mq.dequeue());
49                         System.out.println("Đã xử lý");
50                     }
51                     case 3 -> {
52                         System.out.println("Tin nhắn đã nhận được là :");
53                         ms.display();
54                         break OUTER;
55                     }
56                     case 4 -> mq.display();
57                     default -> {
58                     }
59                 }
60             }
61         } catch (Exception ex){
62             System.out.println("Nhập phím sai!");
63         }
64     }
65 }
66
67

```

Figure 5 Main program



## 1.2. Flowchart

- **Press 1: enter message**

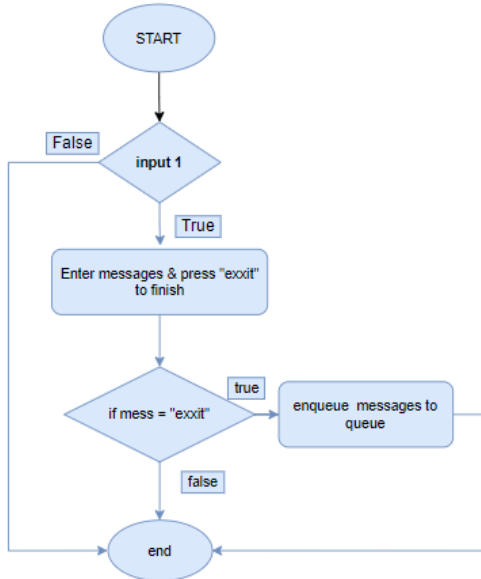


Figure 6 option 1

- **Press 2: send the message**

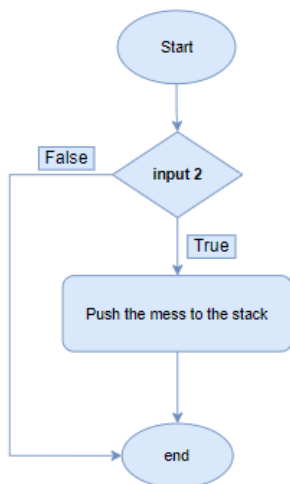


Figure 7 option 2

- Press 3: receive message

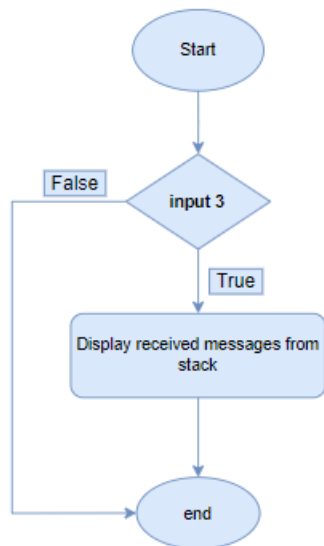


Figure 8 option 3

### 1.3. Execute ADT

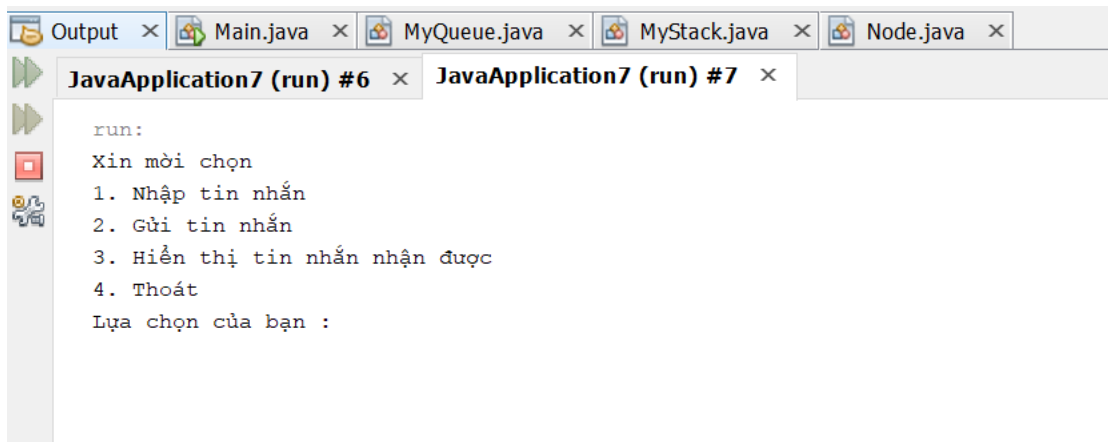
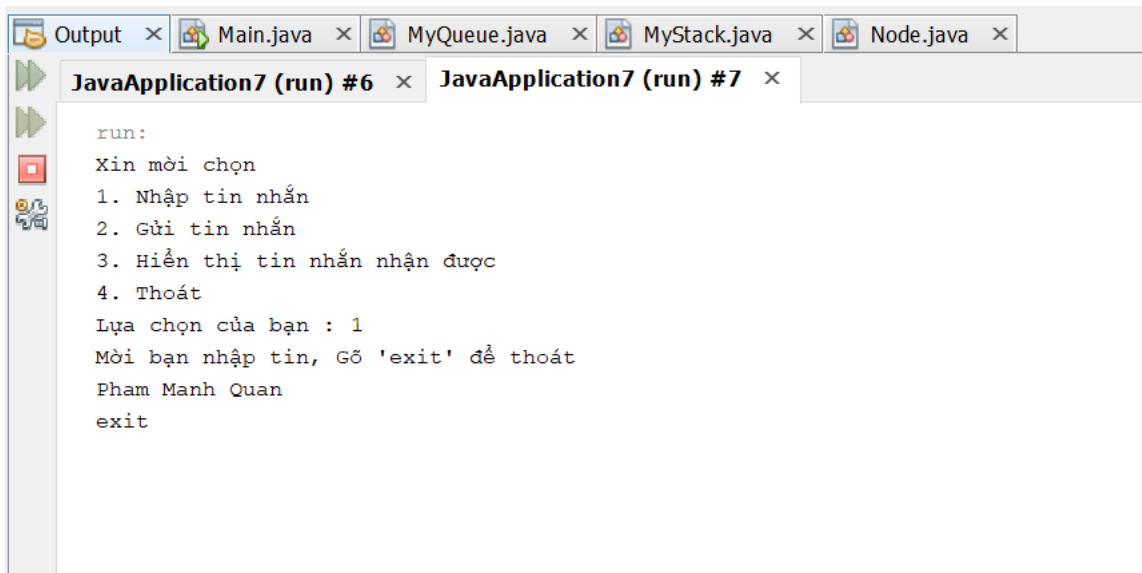


Figure 9 Execute ADT

### ❖ Press 1 to enter message

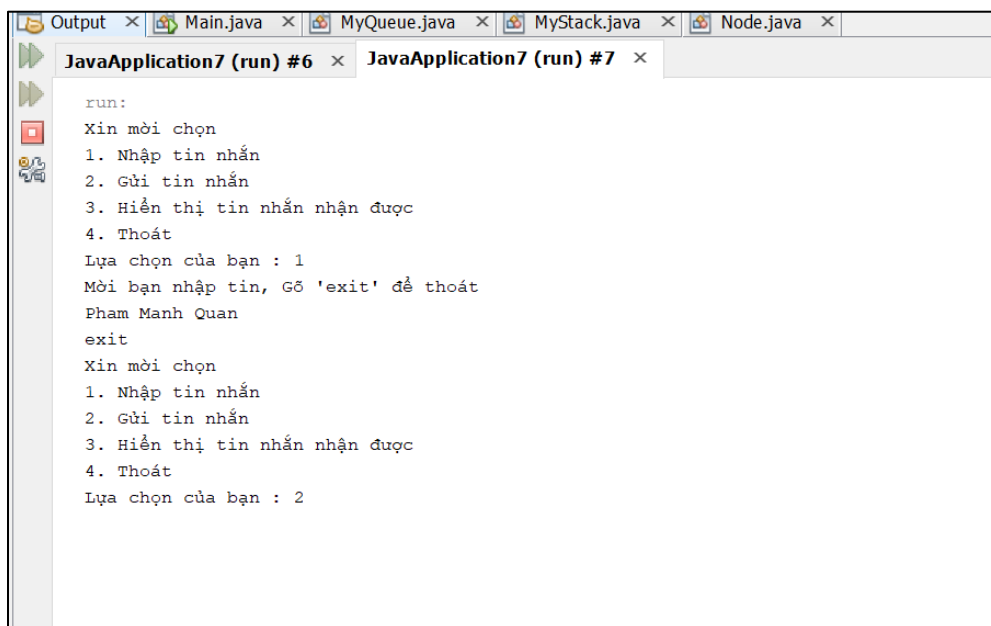


```
run:
Xin moi chon
1. Nhap tin nhan
2. Gui tin nhan
3. Hien thi tin nhan nhan duoc
4. Thoát
Lua chon cua ban : 1
Moi ban nhap tin, Go 'exit' de thoát
Pham Manh Quan
exit
```

Figure 10 Press 1 to enter message

Enter the message: Pham Manh Quan, and the command “exit” to exit

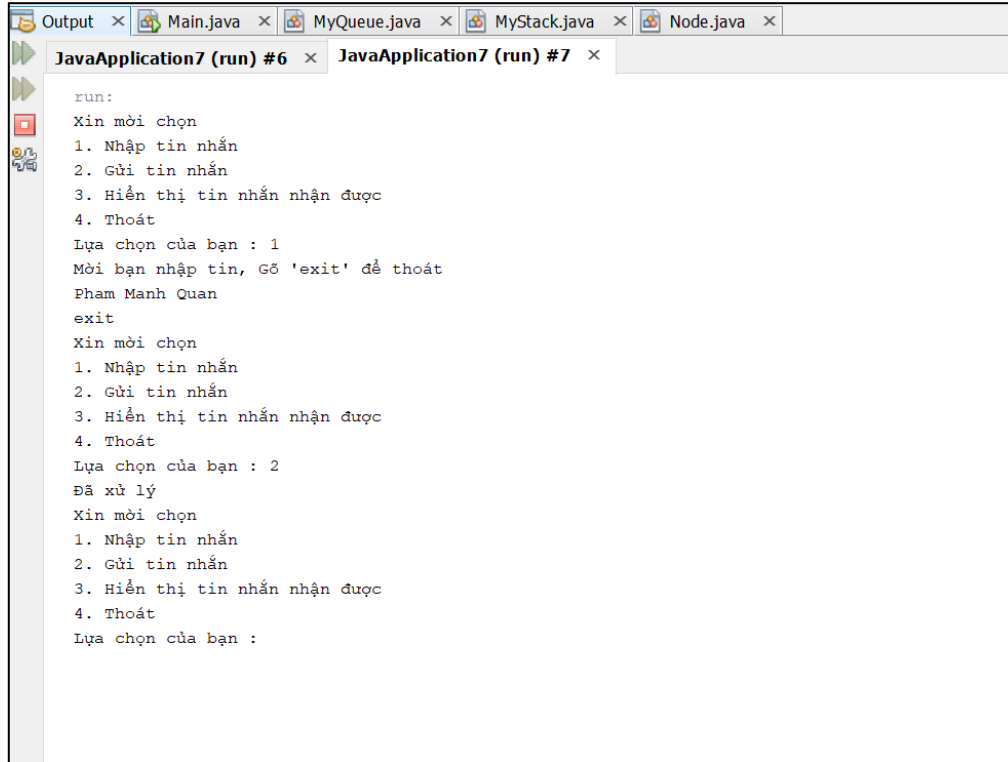
### ❖ Press 2 to send the message



```
run:
Xin moi chon
1. Nhap tin nhan
2. Gui tin nhan
3. Hien thi tin nhan nhan duoc
4. Thoát
Lua chon cua ban : 1
Moi ban nhap tin, Go 'exit' de thoát
Pham Manh Quan
exit
```

Figure 11 Press 2 to send the message

The message has been processed and sent

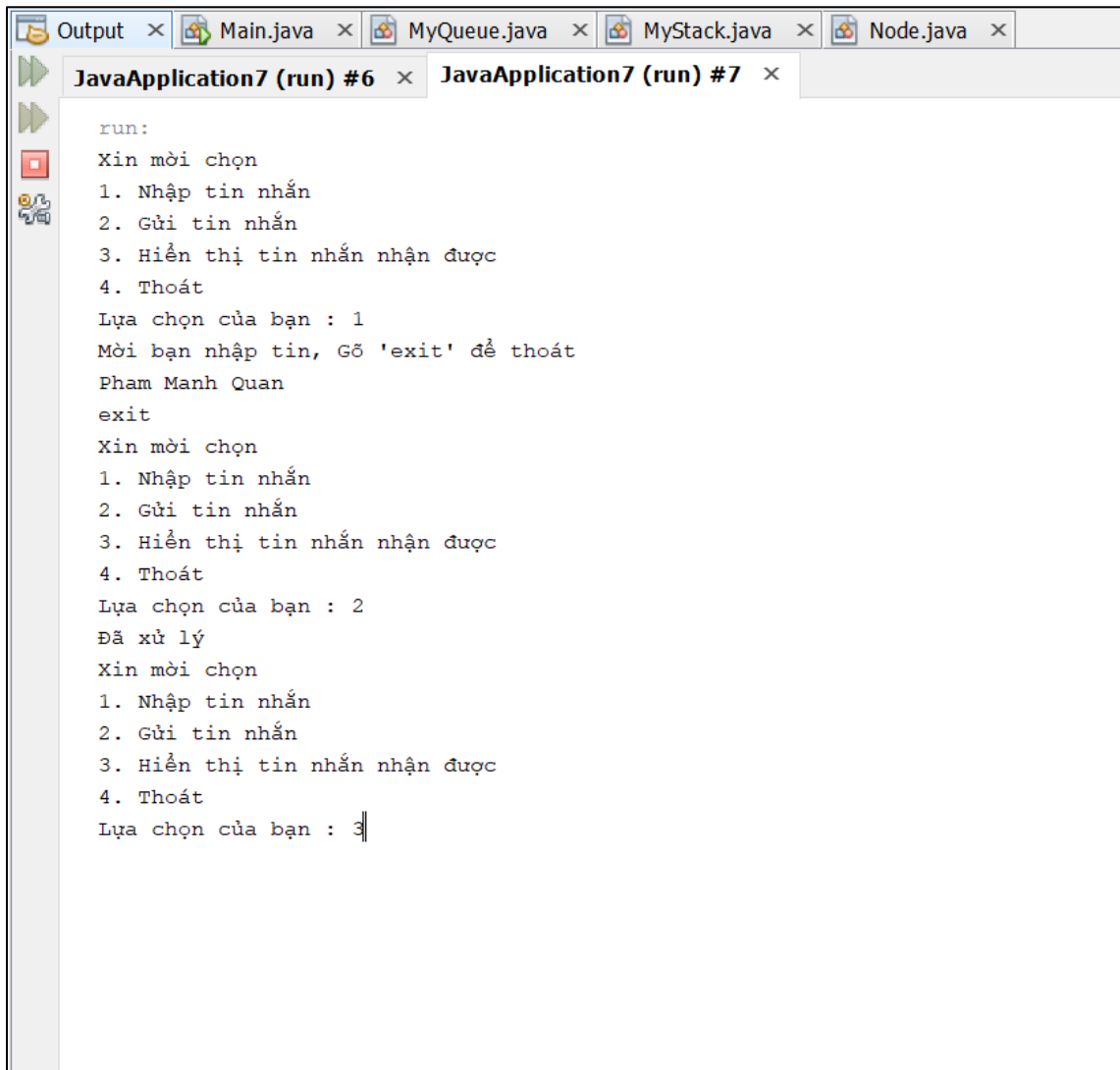


```

run:
Xin mời chọn
1. Nhập tin nhắn
2. Gửi tin nhắn
3. Hiển thị tin nhắn nhận được
4. Thoát
Lựa chọn của bạn : 1
Mời bạn nhập tin, Gõ 'exit' để thoát
Pham Manh Quan
exit
Xin mời chọn
1. Nhập tin nhắn
2. Gửi tin nhắn
3. Hiển thị tin nhắn nhận được
4. Thoát
Lựa chọn của bạn : 2
Đã xử lý
Xin mời chọn
1. Nhập tin nhắn
2. Gửi tin nhắn
3. Hiển thị tin nhắn nhận được
4. Thoát
Lựa chọn của bạn :
  
```

Figure 12 The message has been processed and sent

❖ Press 3 to receive the message

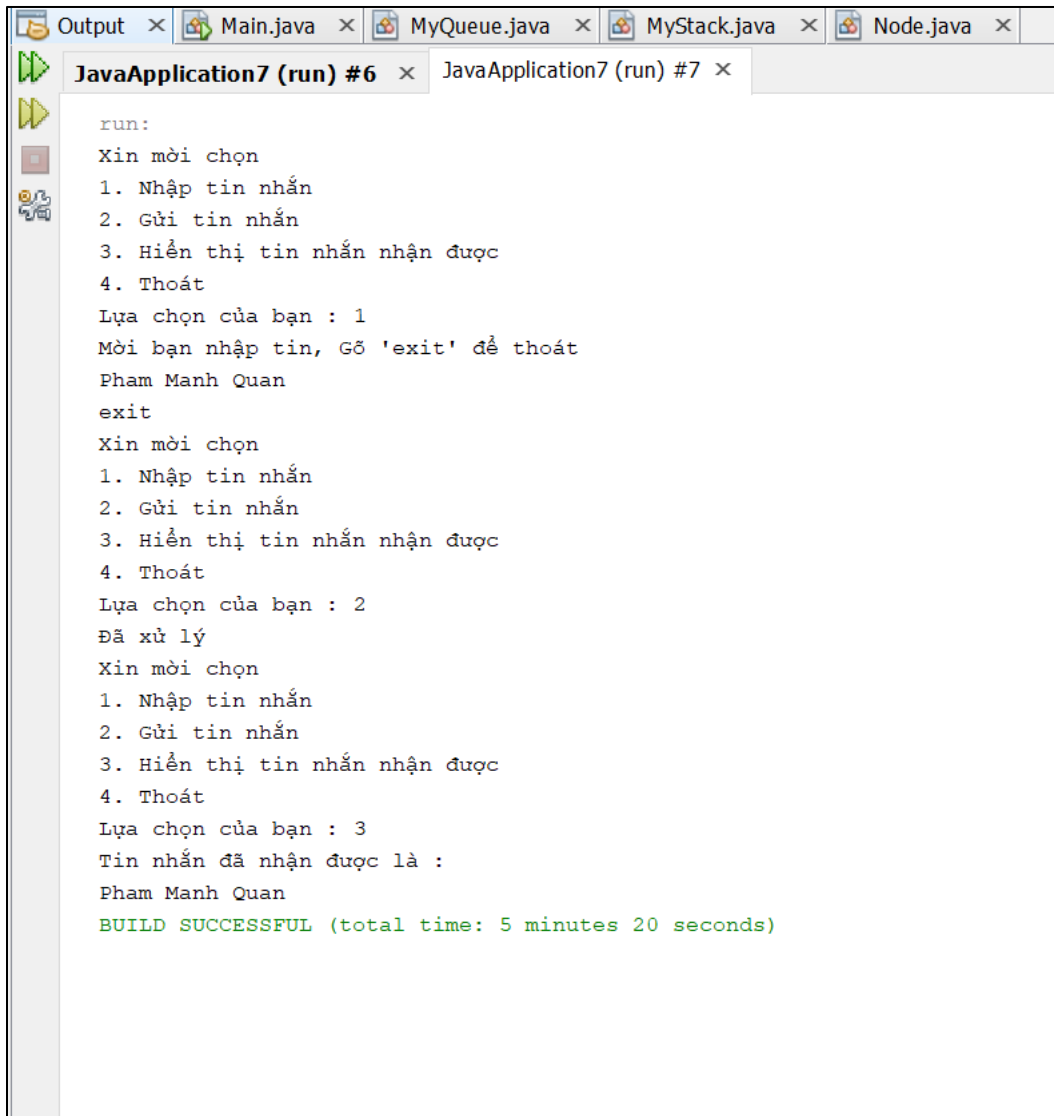


```

Output x Main.java x MyQueue.java x MyStack.java x Node.java x
JavaApplication7 (run) #6 x JavaApplication7 (run) #7 x
run:
Xin mời chọn
1. Nhập tin nhắn
2. Gửi tin nhắn
3. Hiển thị tin nhắn nhận được
4. Thoát
Lựa chọn của bạn : 1
Mời bạn nhập tin, Gõ 'exit' để thoát
Pham Manh Quan
exit
Xin mời chọn
1. Nhập tin nhắn
2. Gửi tin nhắn
3. Hiển thị tin nhắn nhận được
4. Thoát
Lựa chọn của bạn : 2
Đã xử lý
Xin mời chọn
1. Nhập tin nhắn
2. Gửi tin nhắn
3. Hiển thị tin nhắn nhận được
4. Thoát
Lựa chọn của bạn : 3
  
```

Figure 13 Press 3 to receive the message

The program displayed the received message.



```

run:
Xin mời chọn
1. Nhập tin nhắn
2. Gửi tin nhắn
3. Hiển thị tin nhắn nhận được
4. Thoát
Lựa chọn của bạn : 1
Mời bạn nhập tin, Gõ 'exit' để thoát
Pham Manh Quan
exit
Xin mời chọn
1. Nhập tin nhắn
2. Gửi tin nhắn
3. Hiển thị tin nhắn nhận được
4. Thoát
Lựa chọn của bạn : 2
Đã xử lý
Xin mời chọn
1. Nhập tin nhắn
2. Gửi tin nhắn
3. Hiển thị tin nhắn nhận được
4. Thoát
Lựa chọn của bạn : 3
Tin nhắn đã nhận được là :
Pham Manh Quan
BUILD SUCCESSFUL (total time: 5 minutes 20 seconds)

```

Figure 14 the received message

## II. Perform error handling and test results

When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception** (throw an error).

### 2.1. Java try and catch

The **try** statement allows you to define a block of code to be tested for errors while it is being executed.

The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.

The **try** and **catch** keywords come in pairs:

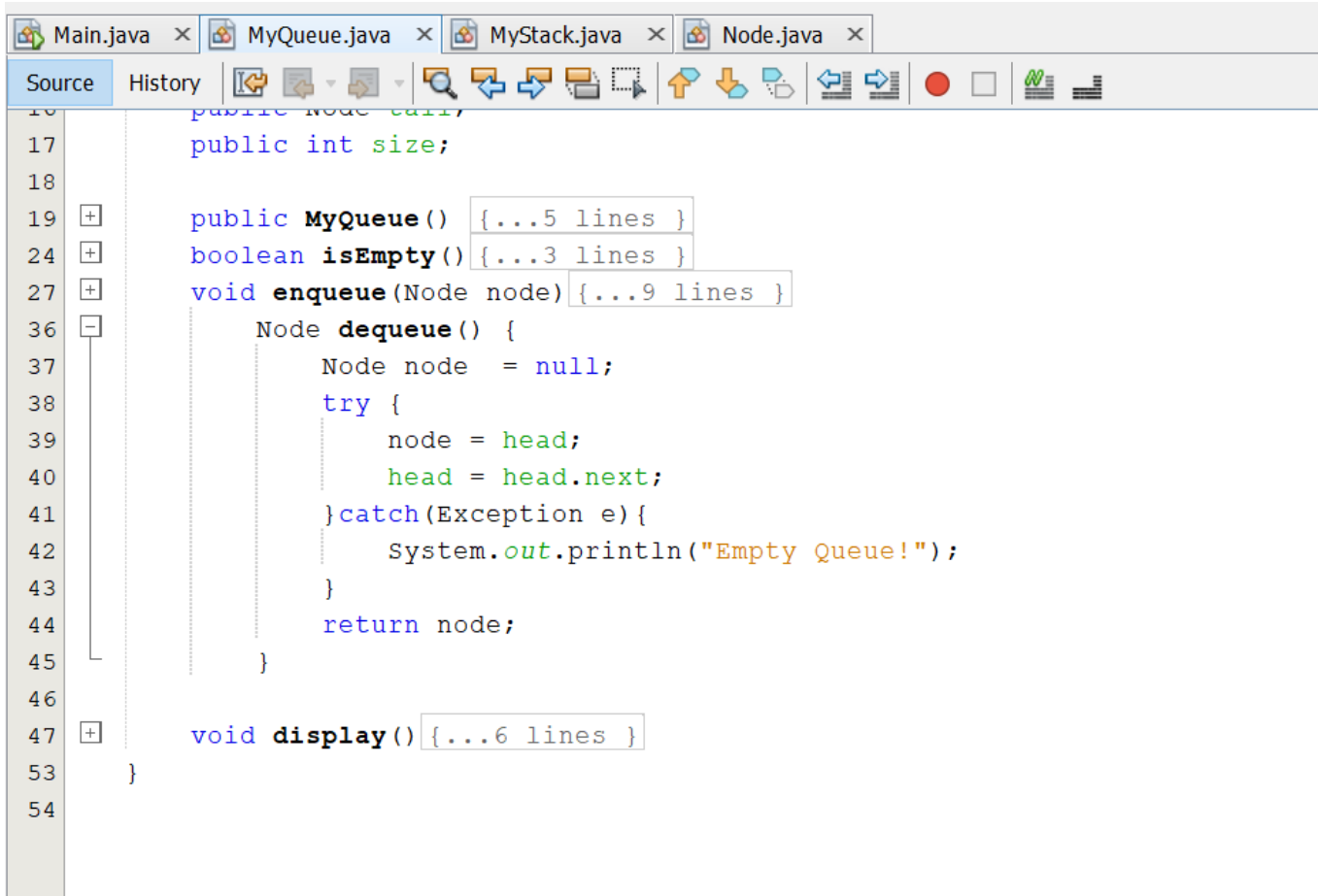
Syntax

```
try {  
    // Block of code to try  
}  
catch(Exception e) {  
    // Block of code to handle errors  
}
```

*Figure 15 Java try and catch*

When the user presses the keys to select the function, the user only allowed to select numbers, if you press a letter, the program will stop and not can keep running If an error occurs, I can use **try...catch** to catch the error and execute some code to handle it

for class MyQueue :



```

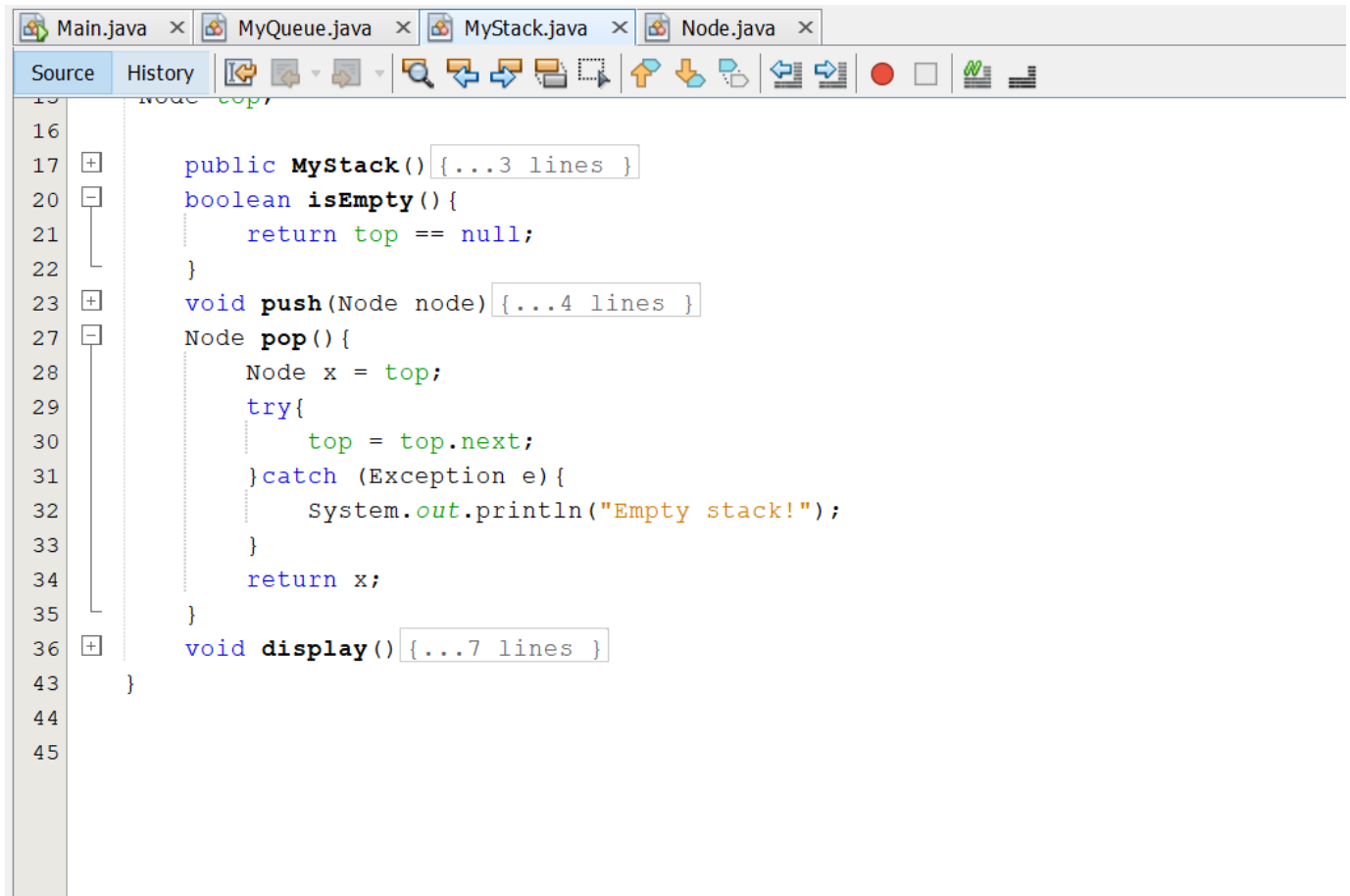
16 public Node tail;
17 public int size;
18
19 public MyQueue() { ...5 lines }
24 boolean isEmpty() { ...3 lines }
27 void enqueue(Node node) { ...9 lines }
36     Node dequeue() {
37         Node node = null;
38         try {
39             node = head;
40             head = head.next;
41         } catch (Exception e) {
42             System.out.println("Empty Queue!");
43         }
44         return node;
45     }
46
47 void display() { ...6 lines }
53 }
54

```

Figure 16 class *MyQueue*



Class MyStack :



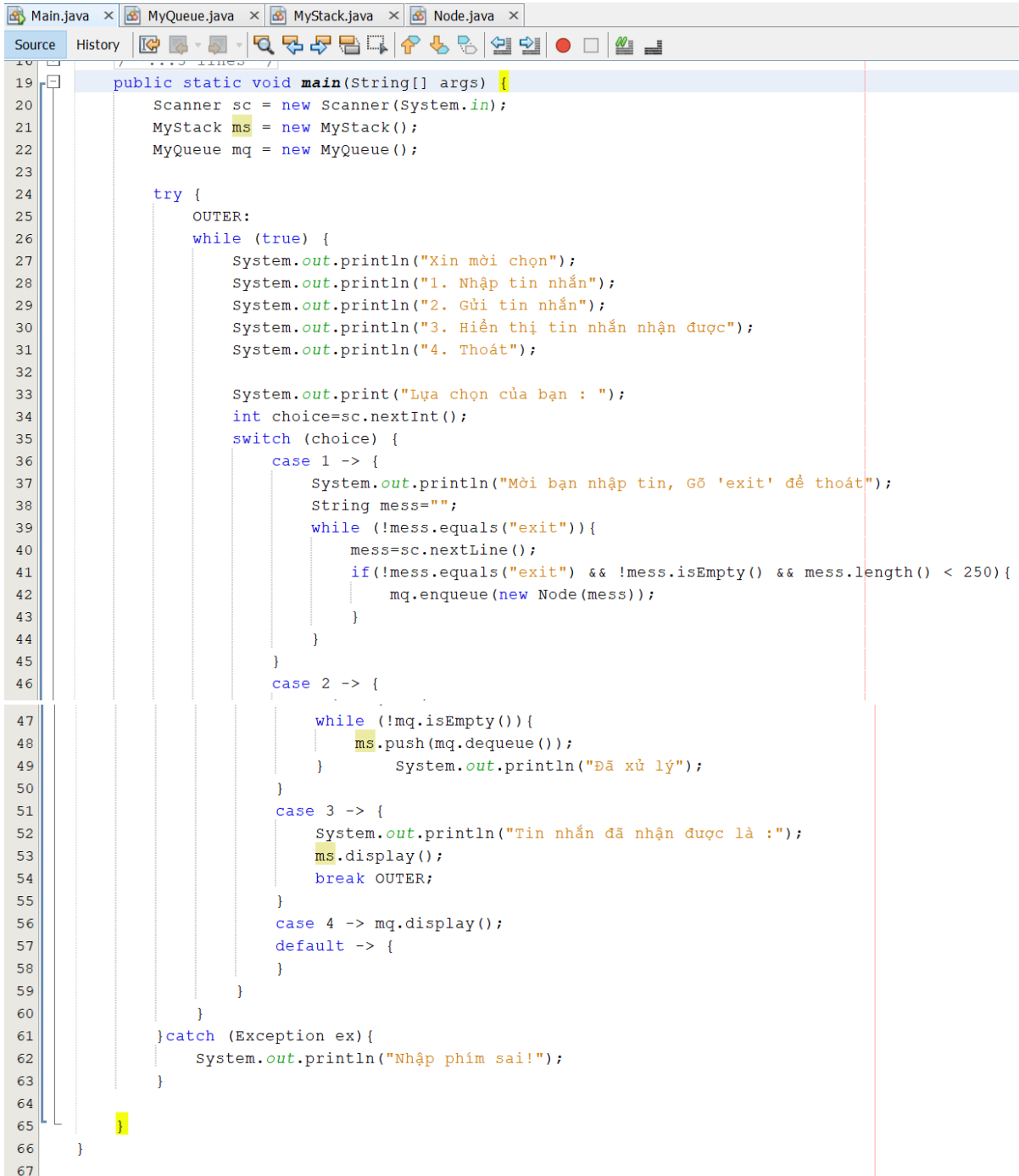
```

16
17 public MyStack() {...3 lines }
20 boolean isEmpty() {
21     return top == null;
22 }
23 void push(Node node) {...4 lines }
27 Node pop() {
28     Node x = top;
29     try{
30         top = top.next;
31     }catch (Exception e){
32         System.out.println("Empty stack!");
33     }
34     return x;
35 }
36 void display() {...7 lines }
43 }
44
45

```

Figure 17 Class MyStack

## Class Main:



```

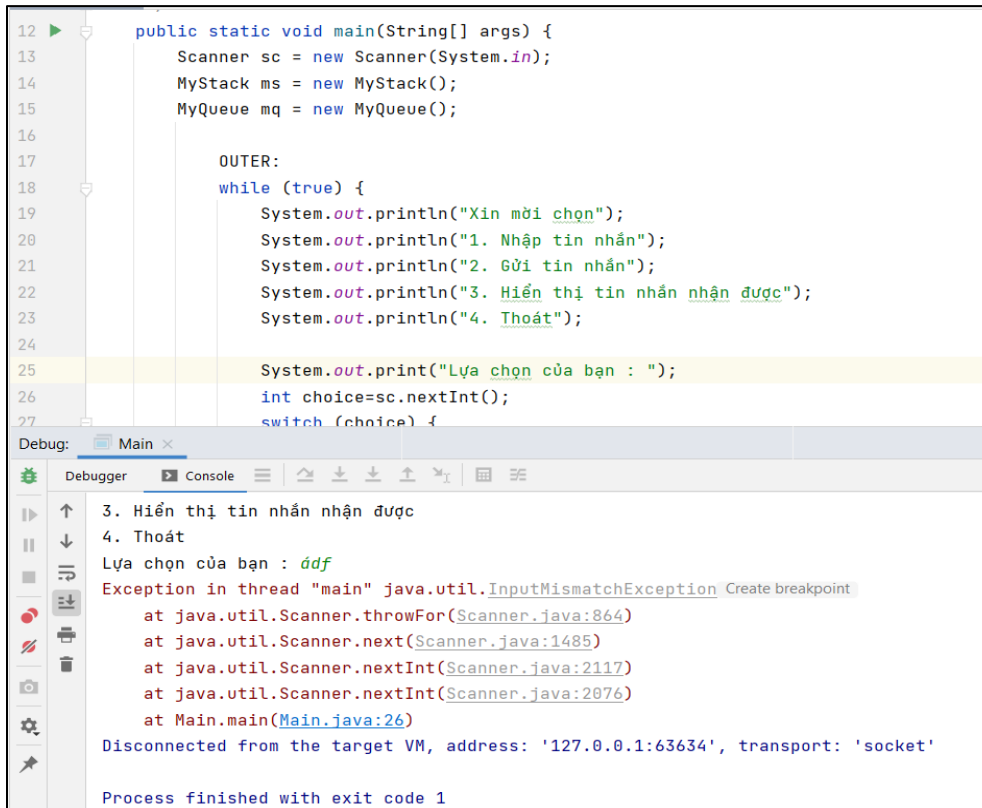
19 public static void main(String[] args) {
20     Scanner sc = new Scanner(System.in);
21     MyStack ms = new MyStack();
22     MyQueue mq = new MyQueue();
23
24     try {
25         OUTER:
26         while (true) {
27             System.out.println("Xin mời chọn");
28             System.out.println("1. Nhập tin nhắn");
29             System.out.println("2. Gửi tin nhắn");
30             System.out.println("3. Hiển thị tin nhắn nhận được");
31             System.out.println("4. Thoát");
32
33             System.out.print("Lựa chọn của bạn : ");
34             int choice=sc.nextInt();
35             switch (choice) {
36                 case 1 -> {
37                     System.out.println("Mời bạn nhập tin, Gõ 'exit' để thoát");
38                     String mess="";
39                     while (!mess.equals("exit")){
40                         mess=sc.nextLine();
41                         if(!mess.equals("exit") && !mess.isEmpty() && mess.length() < 250){
42                             mq.enqueue(new Node(mess));
43                         }
44                     }
45                 }
46                 case 2 -> {
47                     while (!mq.isEmpty()){
48                         ms.push(mq.dequeue());
49                     }
50                     System.out.println("Đã xử lý");
51                 }
52                 case 3 -> {
53                     System.out.println("Tin nhắn đã nhận được là :");
54                     ms.display();
55                     break OUTER;
56                 }
57                 case 4 -> mq.display();
58                 default -> {
59                     }
60             }
61         }catch (Exception ex){
62             System.out.println("Nhập phim sai!");
63         }
64     }
65 }
66 }
67

```

Figure 18 Class Main

## 2.2. try ... catch when execute

### -- Before use try ... catch



```

12 public static void main(String[] args) {
13     Scanner sc = new Scanner(System.in);
14     MyStack ms = new MyStack();
15     MyQueue mq = new MyQueue();
16
17     OUTER:
18     while (true) {
19         System.out.println("Xin mời chọn");
20         System.out.println("1. Nhập tin nhắn");
21         System.out.println("2. Gửi tin nhắn");
22         System.out.println("3. Hiển thị tin nhắn nhận được");
23         System.out.println("4. Thoát");
24
25         System.out.print("Lựa chọn của bạn : ");
26         int choice=sc.nextInt();
27         switch (choice) {

```

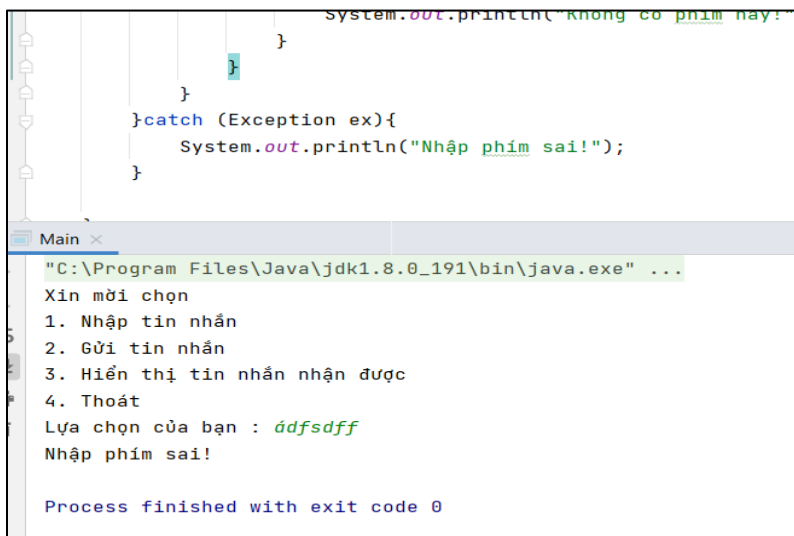
Debugger: Main x

Debugger Console

3. Hiển thị tin nhắn nhận được  
4. Thoát  
Lựa chọn của bạn : adf  
Exception in thread "main" java.util.InputMismatchException Create breakpoint  
at java.util.Scanner.throwFor(Scanner.java:864)  
at java.util.Scanner.next(Scanner.java:1485)  
at java.util.Scanner.nextInt(Scanner.java:2117)  
at java.util.Scanner.nextInt(Scanner.java:2076)  
at Main.main(Main.java:26)  
Disconnected from the target VM, address: '127.0.0.1:63634', transport: 'socket'  
Process finished with exit code 1

Figure 19 Before use try ... catch

### ■ After use try ... catch



```

        System.out.println("Không có phím này!");
    }
}
} catch (Exception ex){
    System.out.println("Nhập phím sai!");
}
}

```

Main x

"C:\Program Files\Java\jdk1.8.0\_191\bin\java.exe" ...  
Xin mời chọn  
1. Nhập tin nhắn  
2. Gửi tin nhắn  
3. Hiển thị tin nhắn nhận được  
4. Thoát  
Lựa chọn của bạn : adfsdff  
Nhập phím sai!  
Process finished with exit code 0

Figure 20 After use try ... catch

### 2.3. Test case

Module test	Enter message, send message, display message
Tester	Pham Manh Quan
Create Date	19/08/2021
Test environment	IDE NetBeans

TEST	WHAT IS BEING TESTED	TEST DATA USED	EXPECTED RESULTS	DATE	ACTUAL RESULT	NOTE
1	Press 1 - Enter message less than 250 word	String = "Pham Manh Quan"	The program can transfer this message	08/19 <sup>th</sup> , 2021	The program treats this input as a text message and sends it	
2	Press 2 - Send the messages	String	Messages will be processed successfully	08/19 <sup>th</sup> , 2021	Messages have been processed and sent successfully	
3	Press 3 - Test message display	String	The messages will be displayed	08/19 <sup>th</sup> , 2021	The messages are all displayed successfully	
4	Use try ... catch to catch the exception		The program will stop and display an error when incorrect input	08/19 <sup>th</sup> , 2021	The program gives the error "Nhập phím sai" and stops program	

### III. The asymptotic analysis technique is used to evaluate the efficiency of the algorithm

#### 3.1 What is Algorithmic Analysis?

Parsing an algorithm is basically counting the number of basic operations that the algorithm performs. Defining exactly what a basic operation is is not trivial. However, for simplicity, we temporarily consider the basic operations here as: addition, subtraction, multiplication, division, comparison and each of these basic operations takes 1 unit of time. Therefore, we sometimes also consider the number of basic operations as a rough estimate of the computation time. I say rough estimation because real time depends a lot on the computer (or computational model) we use. Same 1 million 32-bit multipliers but different hardware's computation time may be different. However, we still accept this rough estimation because it will make the algorithm analysis simpler, removing hardware dependence from the analysis.



*Figure 21 Algorithmic Analysis*

From the point of view of data structures, here are some important algorithms:

Search Algorithm: Algorithm to find an element in a data structure.

Sorting Algorithm: Algorithm to sort the elements in a certain order.

Insertion Algorithm: Algorithm to insert words into a data structure.

Update Algorithm: Algorithm to update (or update) an existing element in a data structure.

Deletion Algorithm: Algorithm to delete an existing element from a data structure.

### 3.2 What is asymptotic analysis?

Asymptotic analysis is asymptotic to input data (Input), ie if the algorithm has no Input, the final conclusion will be that the algorithm will run for a specific amount of time and is constant. In addition to the Input factor, other factors are considered constant.

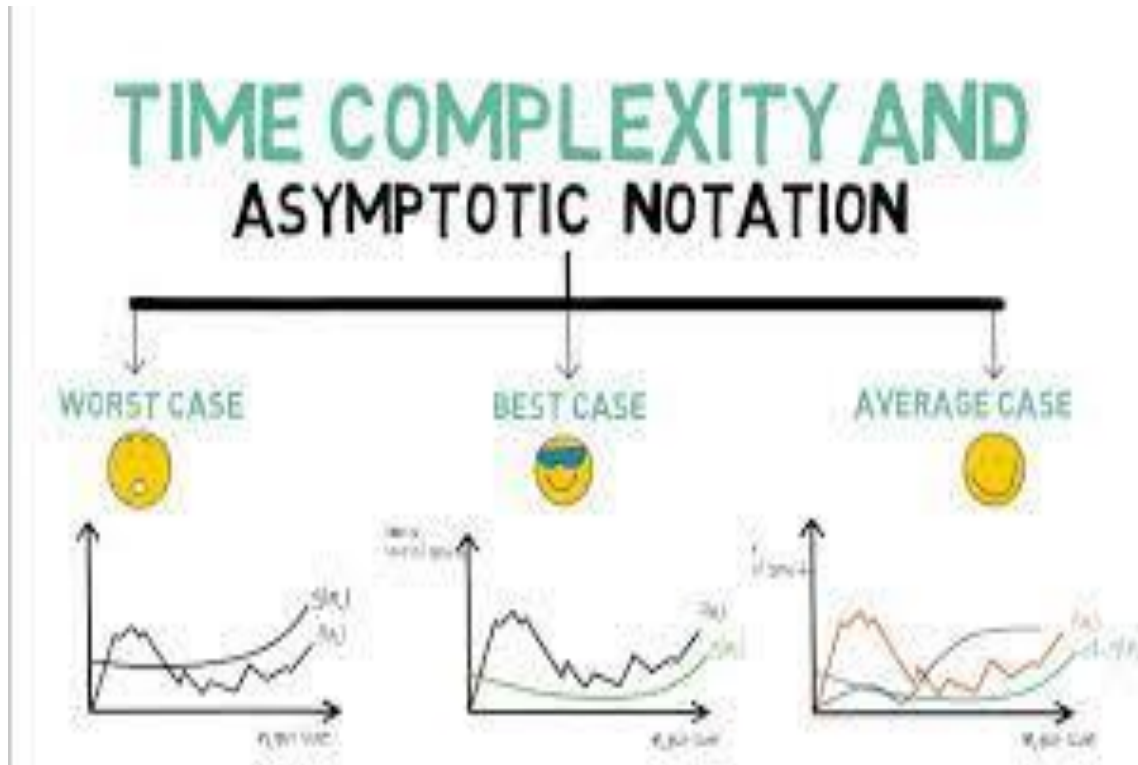


Figure 22 asymptotic analysis

Asymptotic analysis refers to the estimation of the running time of any computation in the computational steps. For example, the running time of a certain calculation is estimated as a function  $f(n)$  and for another calculation as a function  $g(n^2)$ . This means that the running time of the first calculation will increase linearly with the increase of  $n$ , and the running time of the second calculation will increase exponentially as  $n$  increases. Similarly, when  $n$  is quite small, the running time of the two computations is almost the same.

Usually the time required by an algorithm is divided into 3 categories:

- Best case: is the minimum time required to execute the program.
- Average case: is the average time it takes to execute the program.
- Worst case: is the maximum time required to execute the program

#### Asymptotic Notation in Data Structures and Algorithms

O Notation

$\Omega$  Notation

$\theta$  Notation

## Asymptotic Notations

**$\Theta$  Notation:** The theta notation bounds a function from above and below, so it defines exact asymptotic behavior.

A simple way to get Theta notation of an expression is to drop low order terms and ignore leading constants. For example, consider the following expression.

$$3n^3 + 6n^2 + 6000 = \Theta(n^3)$$

Dropping lower order terms is always fine because there will always be a number( $n$ ) after which  $\Theta(n^3)$  has higher values than  $\Theta(n^2)$  irrespective of the constants involved.

- **Big O Notation:** The Big O notation defines an upper bound of an algorithm, it bounds a function only from above. For example, consider the case of Insertion Sort. It takes linear time in best case and quadratic time in worst case. We can safely say that the time complexity of Insertion sort is  $O(n^2)$ . Note that  $O(n^2)$  also covers linear time.

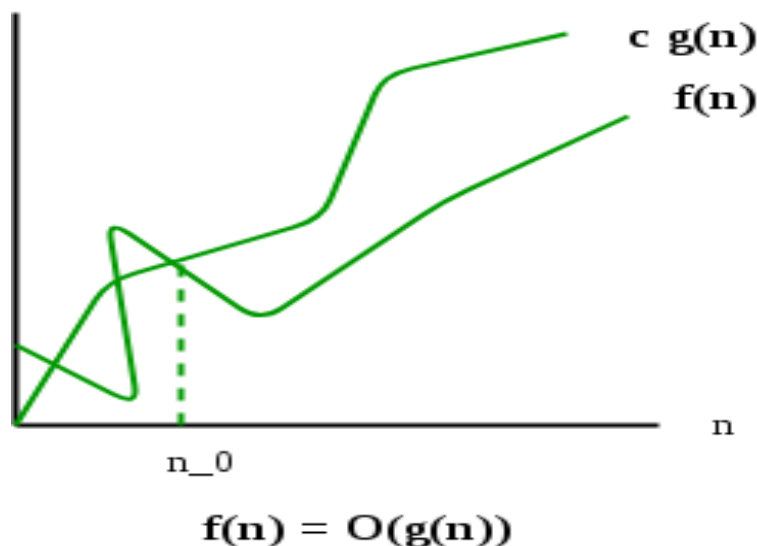


Figure 23 Big O Notation

- The Big O notation is useful when we only have upper bound on time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm.

**Notation  $\Omega$ :** Just as Big O notation provides an upper asymptote of a function, the notation  $\Omega$  provides a lower asymptote.

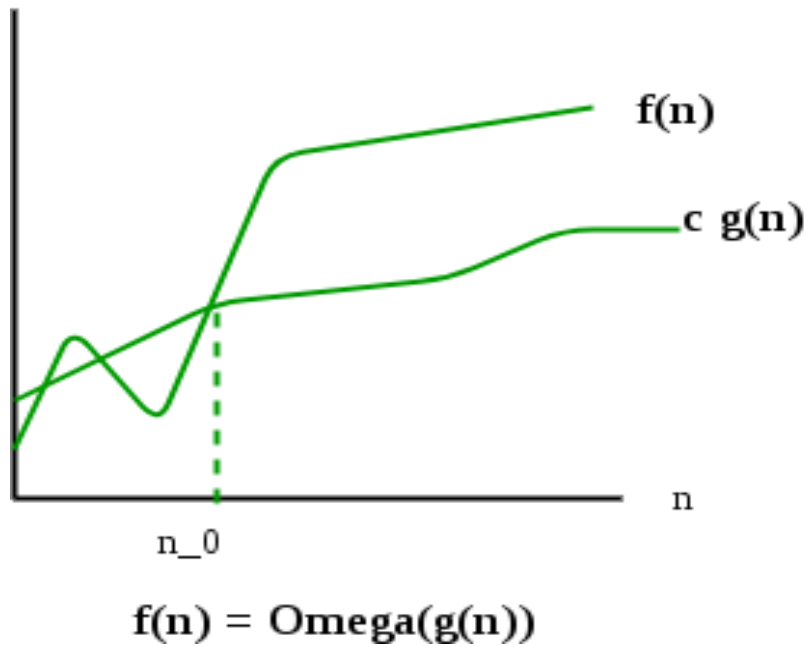


Figure 24 Notation  $\Omega$

$\Omega$  Notation can be useful when we have a lower bound on the time complexity of an algorithm. As discussed in the previous post, the best case performance of an algorithm is often not useful, Omega notation being the least used of all three.

- For example:

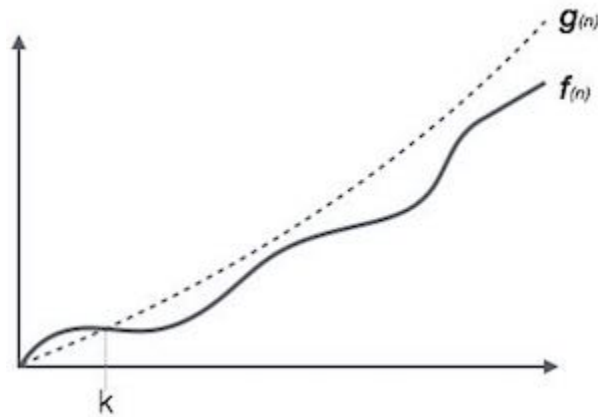
$$\begin{aligned} \blacksquare \quad f(n) &= 8n^2 + 2n - 3 \geq 8n^2 - 3 \\ &= 7n^2 + (n^2 - 3) \geq 7n^2 \quad (g(n)) \end{aligned}$$

Thus,  $k_1 = 7$

### 3.3 Big Oh Notation, in Data Structures and Algorithms

$O(n)$  is a way to represent the upper asymptote of the running time of an algorithm. It estimates the worst case time complexity or the longest amount of time required by an algorithm (execution from start to finish). The graph shows the following:



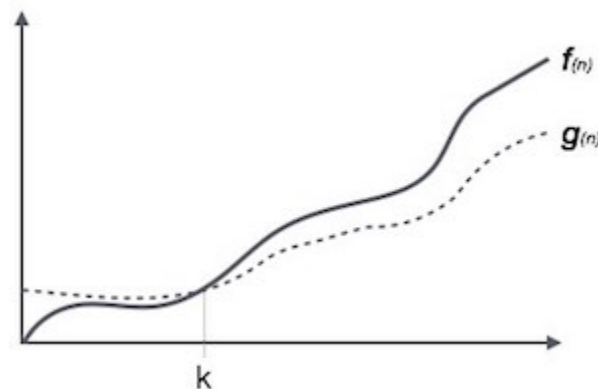


For example, calling  $f(n)$  and  $g(n)$  non-reducing functions defined on positive integers (all time functions satisfy these conditions):

$O(f(n)) = \{ g(n) : \text{if there exist } c > 0 \text{ and } n_0 \text{ such that } g(n) \leq c \cdot f(n) \text{ for all } n > n_0. \}$

### 3.4 Omega Notation, in Data Structures and Algorithms

The  $\Omega(n)$  is a way to represent the lower asymptote of the running time of an algorithm. It estimates the best case time complexity or the shortest amount of time required by an algorithm. The graph shows the following:



For example, for a function  $f(n)$ :

$\Omega(f(n)) = \{ g(n) : \text{if there exist } c > 0 \text{ and } n_0 \text{ such that } g(n) \geq c \cdot f(n) \text{ for all } n > n_0. \}$

### 3.5 Theta Notation, in Data Structures and Algorithms

The  $\theta(n)$  is a way to represent both the upper and lower asymptotes of the running time of an algorithm. You look at the map then:

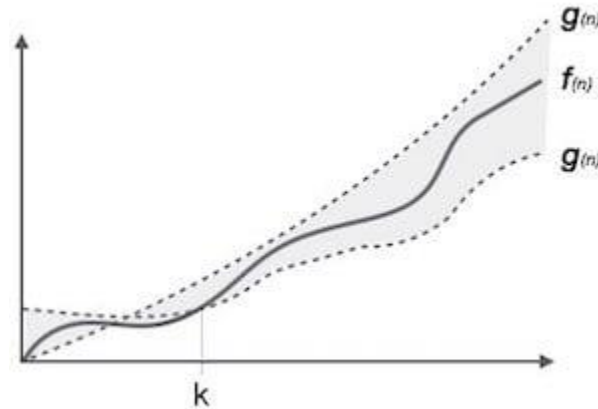


Figure 25 Theta Notation

Some popular Asymptotic Notations in data structures and algorithms

constant –  $O(1)$

logarithm –  $(\log n)$

Linear (Linear) –  $O(n)$

$n \log n$  –  $O(n \log n)$

Quadratic –  $O(n^2)$

Degree 3 (cubic) –  $(n^3)$

Polynomial –  $nO(1)$

Exponential –  $2O^n$

## IV. Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.

### 4.1. What is space complexity?

#### a. Definition

The space complexity of an algorithm or a computer program is the amount of memory space required to solve an instance of computational problem as a function of the size of input.

Similar to time complexity, space complexity is often expressed asymptotically in big O notation, such as  $O(n)$ ,  $O(n \log(n))$  ... where **n** is the input size in units of bits needed to represent the input

# Time & Space Complexity



Figure 26 space complexity

**For example**, if we want to compare standard sorting algorithms based on space, then Auxiliary Space would be a better criterion than Space Complexity. Merge Sort uses  $O(n)$  auxiliary space, Insertion sort, and Heap Sort uses  $O(1)$  auxiliary space. The space complexity of all these sorting algorithms is  $O(n)$  though.

Space complexity is a parallel concept to time complexity. If we need to create an array of size  $n$ , this will require  $O(n)$  space. Creating a two-dimensional array of size  $n*n$  will require  $O(n^2)$  space.

## b. Example

Example with simple algorithm: Space complexity is:  $O(1)$

```
public static void main(String[] args) {  
  
    int x = 10; // 4 bytes  
    int y = 10; // 4 bytes  
    int z = x+y; // 4 bytes  
    System.out.println("Result = " + z); // 4 bytes  
}
```

Figure 27 Example with simple algorithm

Example with for loop: Space complexity is:  $O(n)$

```
static int sumOfArray(int[] arr){    // arr → N*4 bytes
    int sum = 0;                    // sum → 4 bytes
    for (int j : arr) {             // j → 4 bytes
        sum += j;                   // auxiliary → 4 bytes
    }
    return sum;                     // total = 4 * N + 12
}
```

*Figure 28 Example with for loop*

#### 4.2. What is time complexity?

##### a. Definition

**Time complexity** is the amount of time taken by an algorithm to run, as a function of the length of the input. It measures the time taken to execute each statement of code in an algorithm.

##### b. Time Complexity Introduction

Space and Time define any physical object in the Universe. Similarly, Space and Time complexity can define the effectiveness of an algorithm. While we know there is more than one way to solve the problem in programming, knowing how the algorithm works efficiently can add value to the way we do programming. To find the effectiveness of the program/algorithm, knowing how to evaluate them using Space and Time complexity can make the program behave in required optimal conditions, and by doing so, it makes us efficient programmers.

While we reserve the space to understand Space complexity for the future, let us focus on Time complexity in this post. Time is Money! In this post, you will discover a gentle introduction to Time complexity of an algorithm, and how to evaluate a program based on Time complexity.

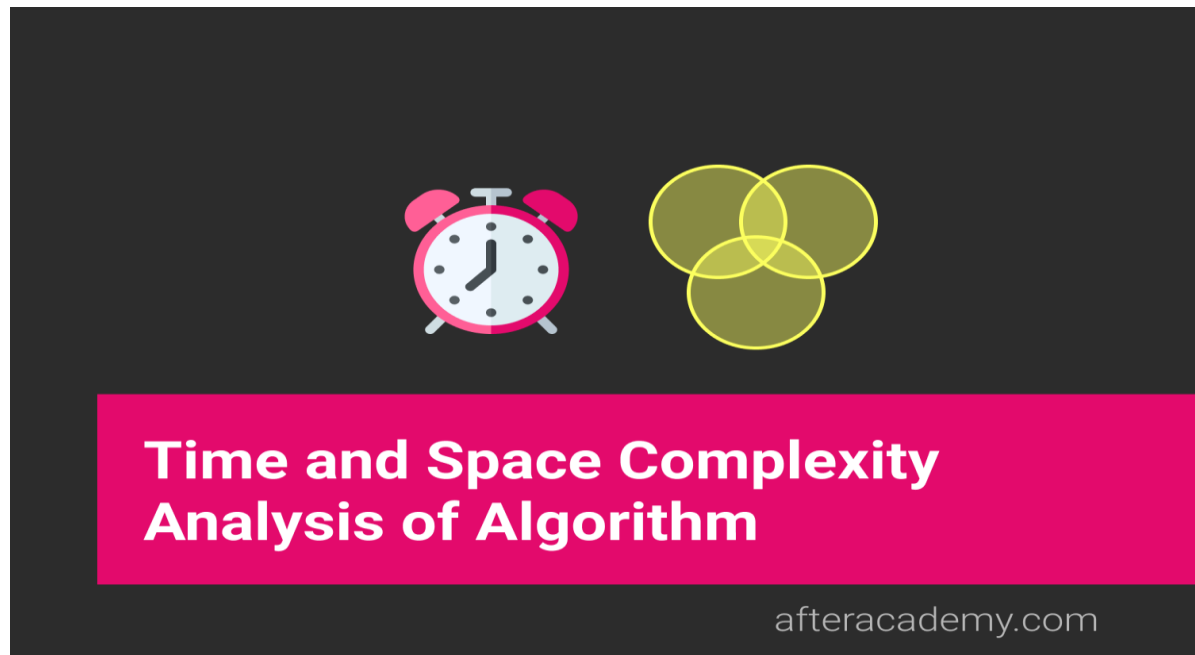


Figure 29 time complexity

### c. Example

Example with simple algorithm: Time complexity is:  $O(1)$

```
public static void main(String[] args) {  
  
    int x = 10; // 1 time  
    int y = 10; // 1 time  
    int z = x+y; // 1 time  
    System.out.println("Result = " + z); // 1 time  
}
```

Figure 30 Example with simple algorithm

Example with for loop: Time complexity is:  $O(n)$

```
int n = 100;           // 1 time
for (int i = 0; i < n; i++){ // n + 1 time
    System.out.println(i); // n time
}
```

Figure 31 Example with for loop

Example with nested loop: Time complexity is:  $O(n^2)$

```
int n = 10; // 1 time
int sum = 0; // 1 time
for (int i = 0; i < n; i++){ // n time
    for (int j = 0; j < n; j++){ // n time
        sum += i; // n * n time
        System.out.println("Result = " + sum); // 1 time
    }
}
```

Figure 32 Example with nested loop

#### d. General Rules for Estimation

**Loops:** The running time of a loop is at most the running time of the statements inside of that loop times the number of iterations.

**Nested Loops:** Running time of a nested loop containing a statement in the inner the most loop is the running time of the statement multiplied by the product of the size of all loops.

**Consecutive Statements:** Just add the running times of those consecutive statements.

**If/Else:** Never more than the running time of the test plus the larger of running times of S1 and S2.

*(Quote: Google classroom)*

## CONCLUDED

To summarize, in this exercise, I worked on complex algorithms and data structures by building a software chat system using stack and queue ADTs. Then I Perform the error handling and report the test results and use the asymptotic analysis technique used to evaluate the algorithm performance.

## REFERENCES

- 1) [www.javatpoint.com](http://www.javatpoint.com). 2021. Exception Handling in Java | Java Exceptions - javatpoint. [online] Available at: [Accessed 11 August 2021].
- 2) Runestone.academy. 2020. 3.3. Big-O Notation — Problem Solving With Algorithms And Data Structures. [online] Available at: [Accessed 1 March 2021].
- 3) Kumer, B., 2020. Space Complexity. [Online] Available at: [Accessed 3 3 2021].
- 4) ResearchGate. 2020. How Can I Measure The Performance Of An Algorithm?. [online] Available at: [Access ed 3 March 2021]
- 5) HackerEarth. 2021. Time and Space Complexity Tutorials & Notes | Basic Programming | HackerEarth. [online] Available at: [Accessed 18 August 2021].