**ASSIGNMENT 2 FRONT SHEET**

|  |  |  |  |
| --- | --- | --- | --- |
| **Qualification** | **BTEC Level 5 HND Diploma in Computing** | | |
| **Unit number and title** | Unit 19: Data Structures and Algorithms | | |
| **Submission date** | August 19th, 2021 | **Date Received 1st submission** | August 19th, 2021 |
| **Re-submission Date** |  | **Date Received 2nd submission** |  |
| **Student Name** | Pham Manh Quan | **Student ID** | BHAF190226 |
| **Class** | BH-AF-2005-2.3 | **Assessor name** | Ngo Thi Mai Loan |
| **Student declaration**  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. | | | |
|  |  | **Student’s signature** |  |

**Grading grid**

|  |  |  |  |
| --- | --- | --- | --- |
| P4 | P5 | P6 | P7 |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
| **❒ Summative Feedback: ❒ Resubmission Feedback:** | | |
| **Grade:** | **Assessor Signature:** | **Date:** |
| **Internal Verifier’s Comments:** | | |
| **IV Signature:** | | |
|  | | |

Contents

[**INTRODUCE** 4](#_Toc80293558)

[**CONTENT** 4](#_Toc80293559)

[**I. Implement a complex ADT and algorithm in an executable programming language to solve a well-defined problem.** 4](#_Toc80293560)

[1.1 ADT design 4](#_Toc80293561)

[1.2 Execute ADT 8](#_Toc80293562)

[**II. Perform error handling and test results** 13](#_Toc80293563)

[Java try and catch 13](#_Toc80293564)

[**III. The asymptotic analysis technique is used to evaluate the efficiency of the algorithm** 17](#_Toc80293565)

[3.1 What is Algorithmic Analysis? 17](#_Toc80293566)

[3.2 What is asymptotic analysis? 18](#_Toc80293567)

[3.3 Big Oh Notation, in Data Structures and Algorithms 20](#_Toc80293568)

[3.4 Omega Notation, in Data Structures and Algorithms 21](#_Toc80293569)

[3.5 Theta Notation, in Data Structures and Algorithms 22](#_Toc80293570)

[**IV. Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.** 22](#_Toc80293571)

[4.1. What is space complexity? 22](#_Toc80293572)

[**CONCLUDED** 24](#_Toc80293573)

[**REFERENCES** 24](#_Toc80293574)

# **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.

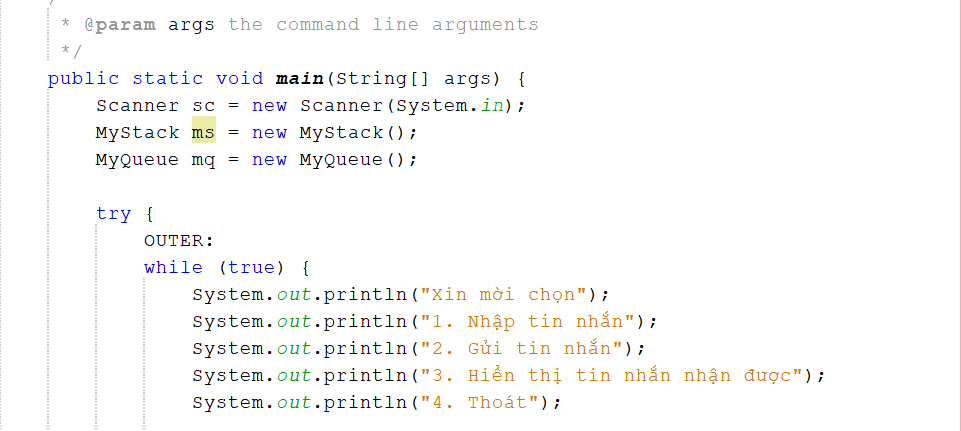
# **CONTENT**

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

### **1.1 ADT design**

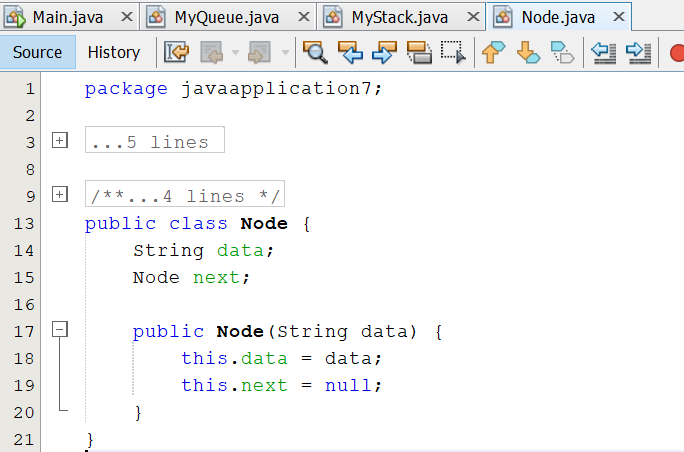
In the article, I discussed the following options:

* 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

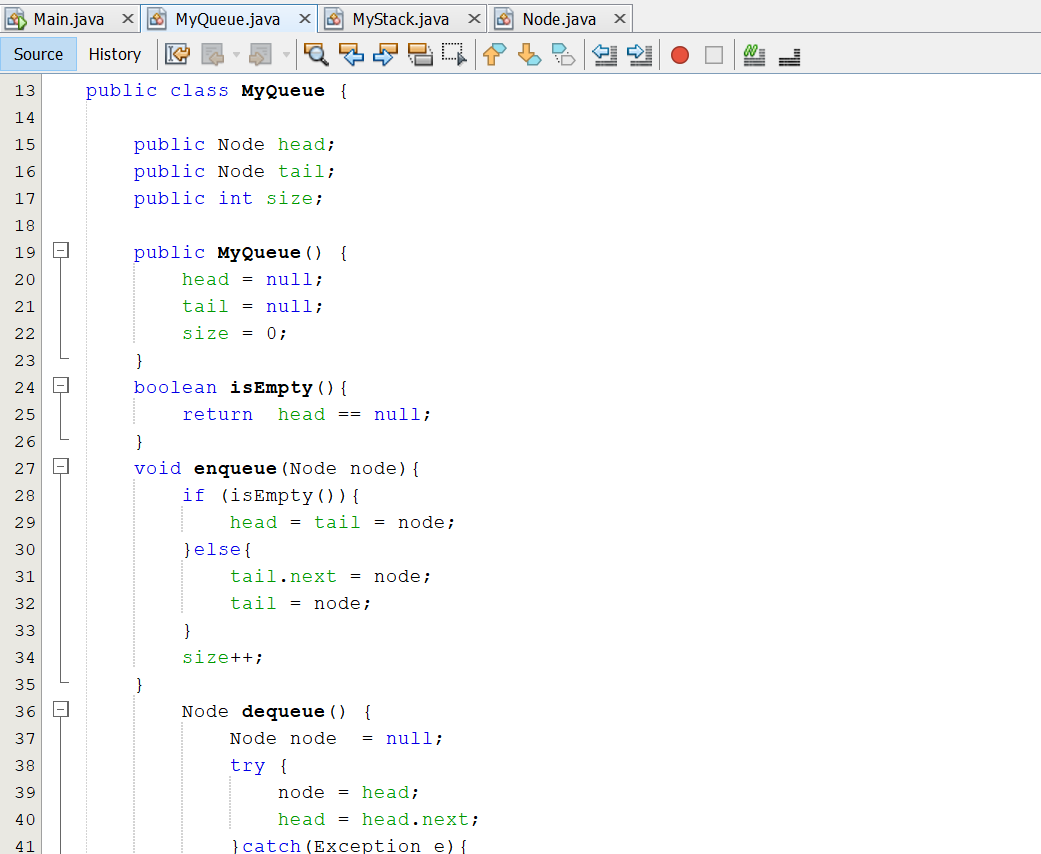


I will use Linked list to design for Node class and MyQueue class. So all messages will be stored in class MyQueue.

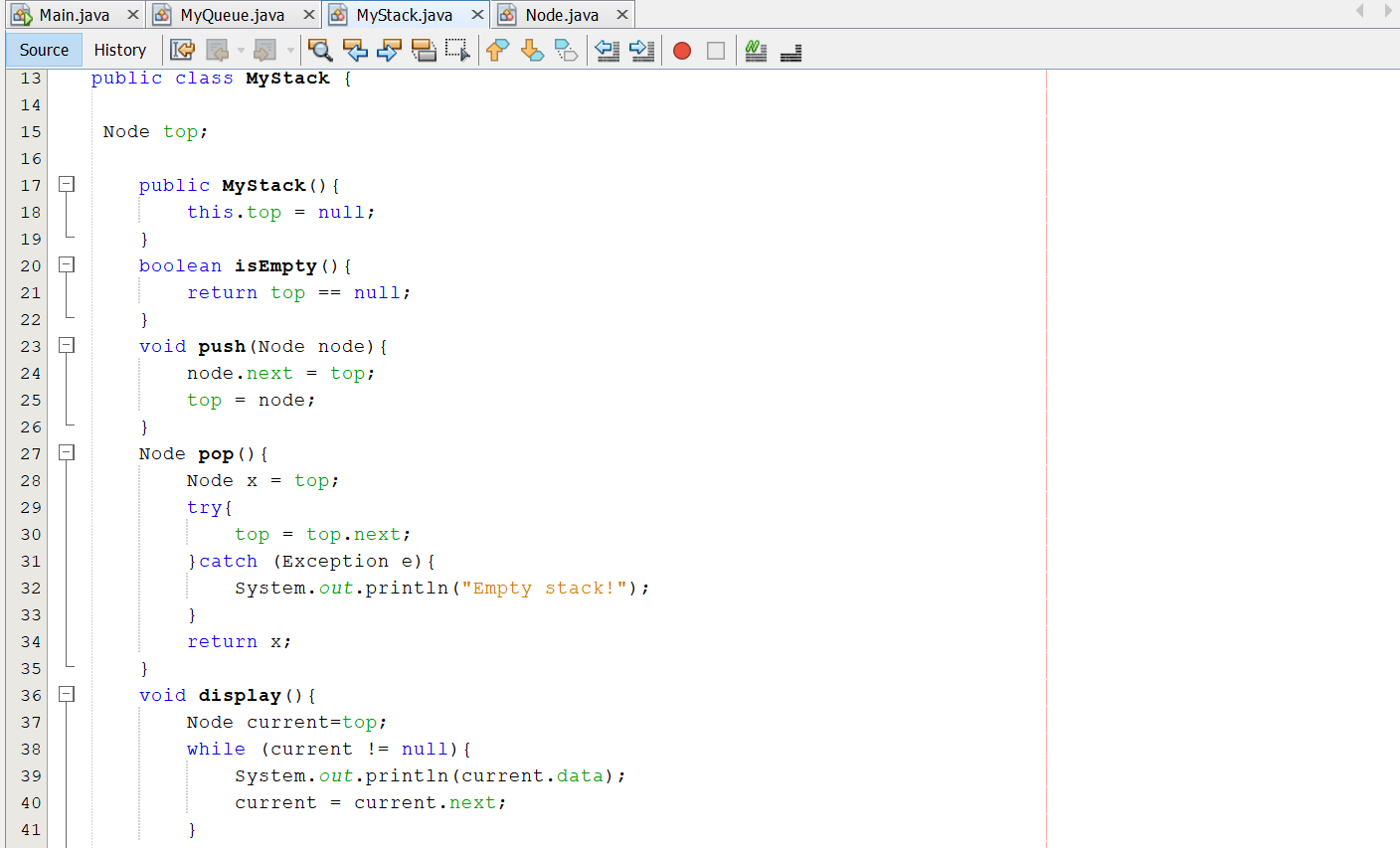
* Class Node



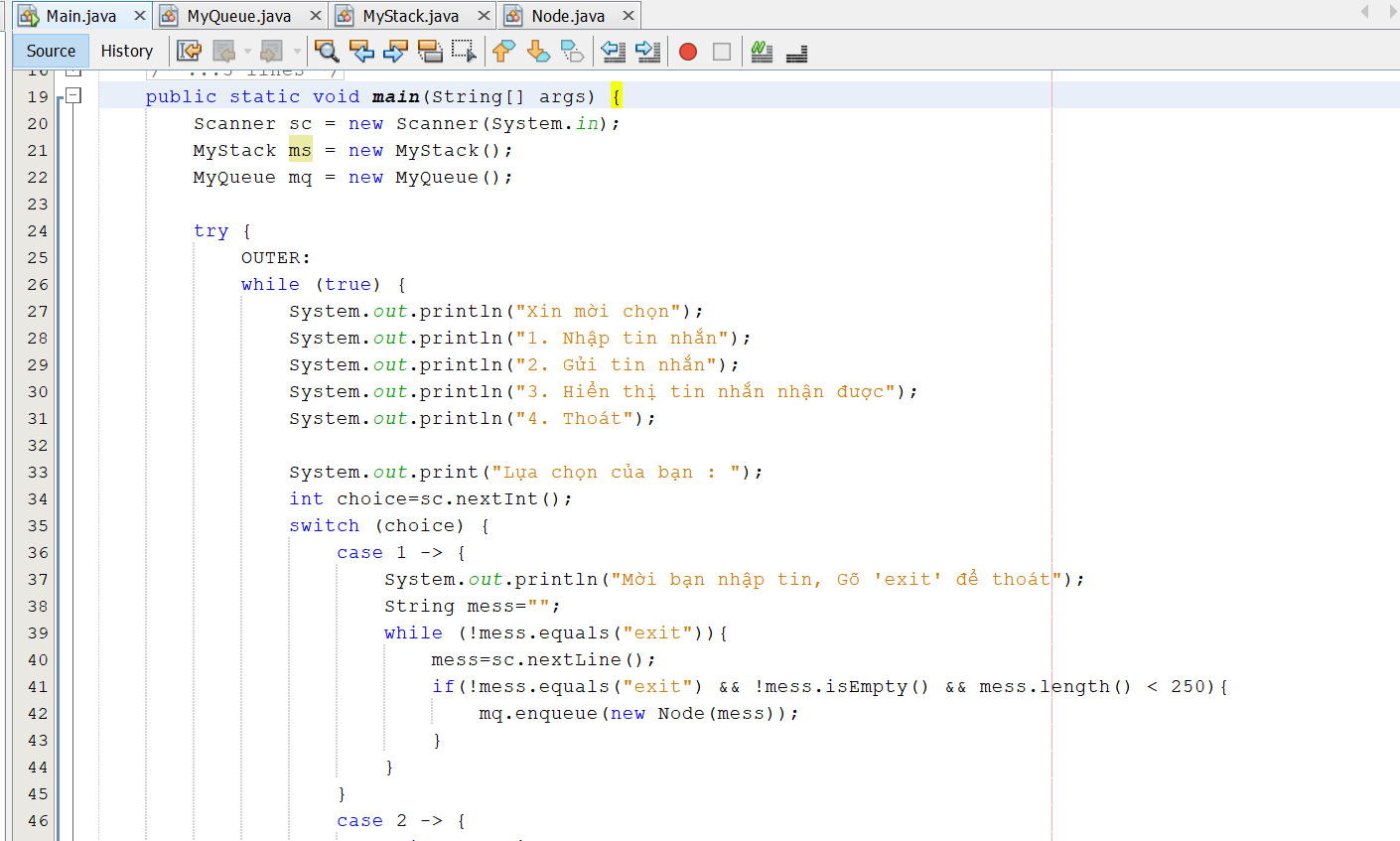
* Class MyQueue



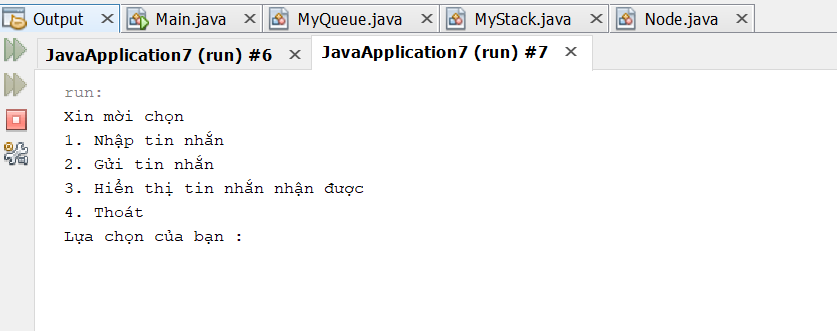
- Class MyStack



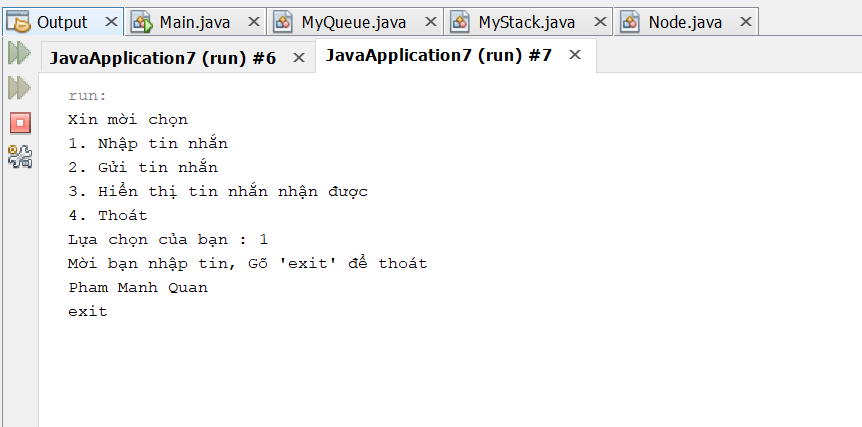
And finally, the code of the main program



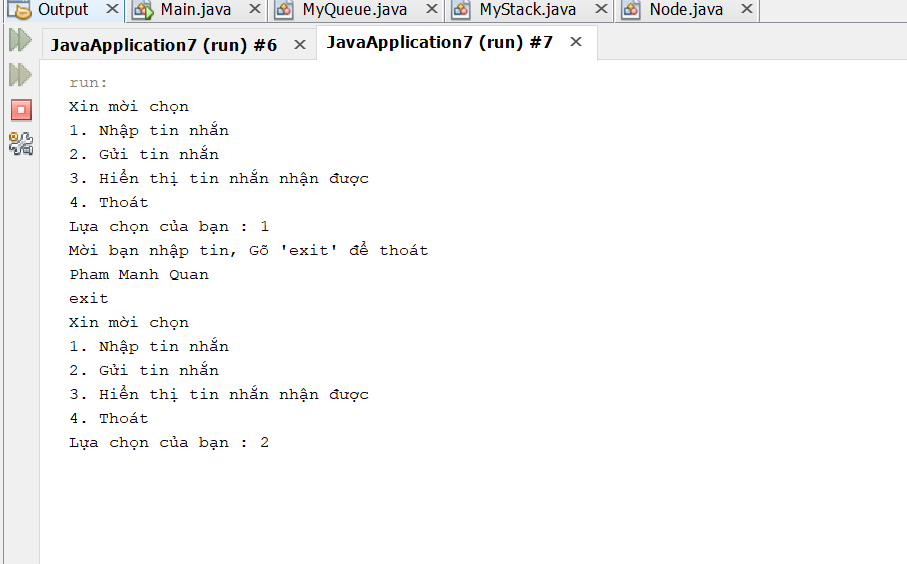
### 1.2 Execute ADT

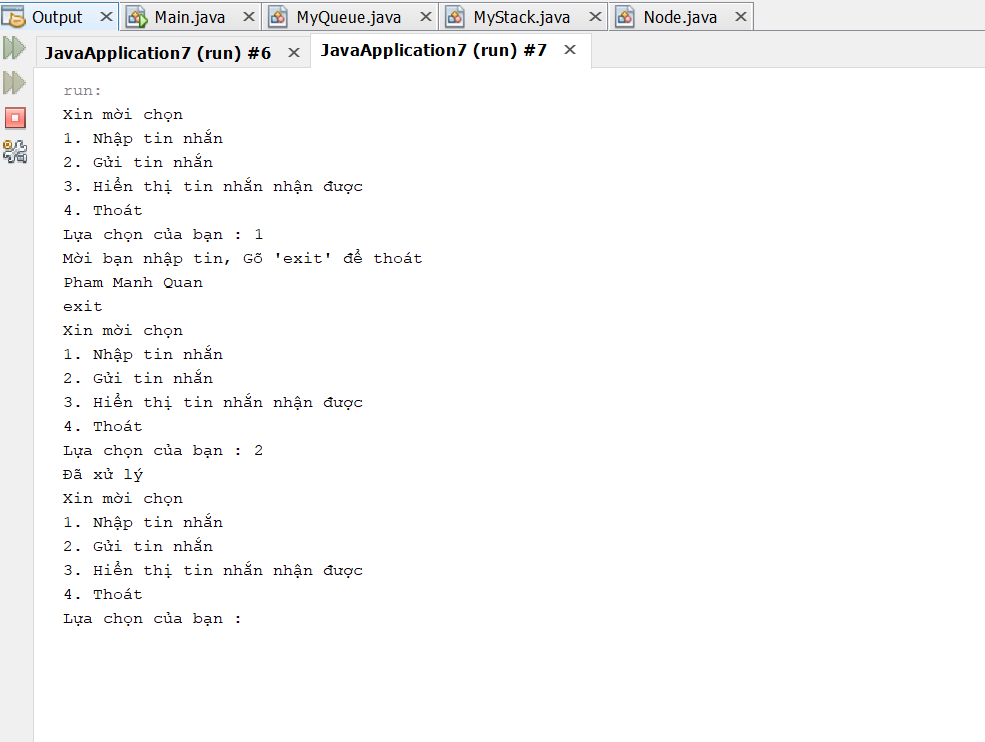


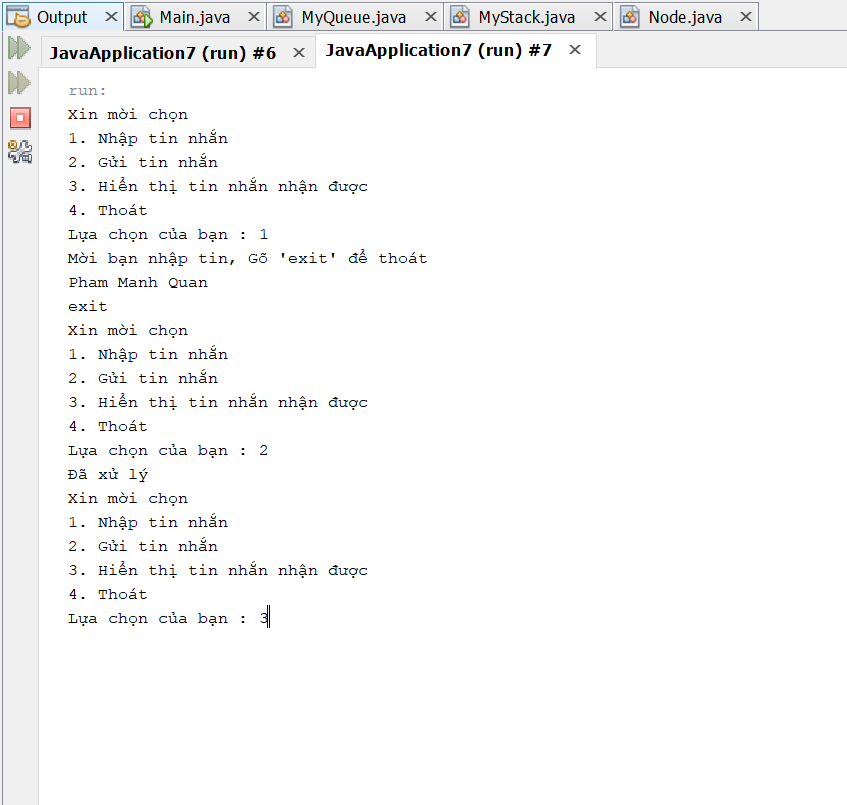
1. enter a message



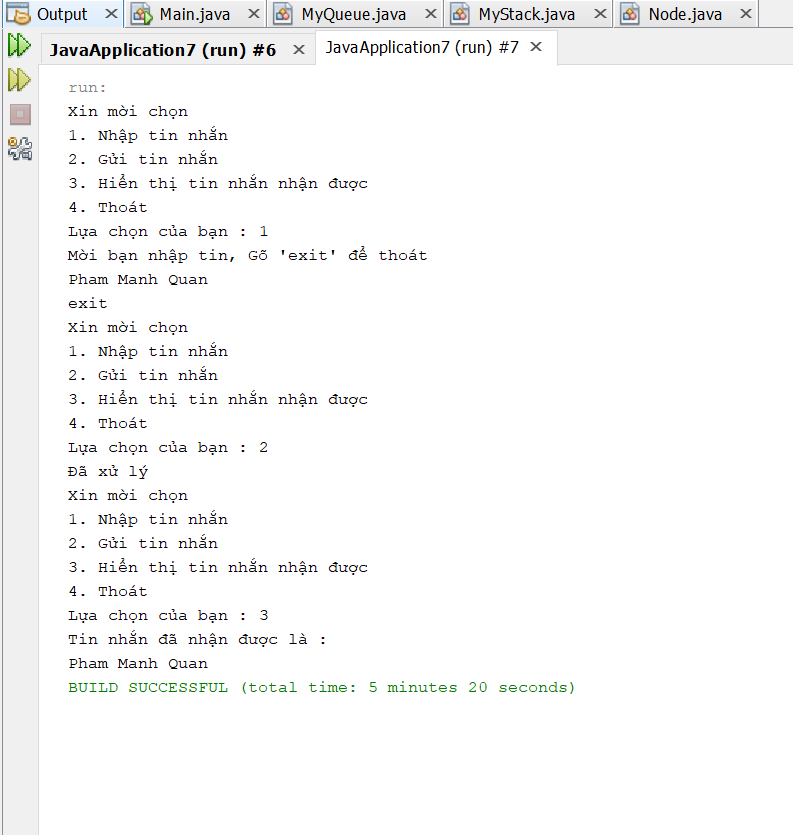
enter the message: Pham Manh Quan , and the command ( exit ) to exit



Enter 2 to send a message has been processed and sent



Enter ( 3 ) to display received messages



the message has been received successfully, the screen appears

## **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).

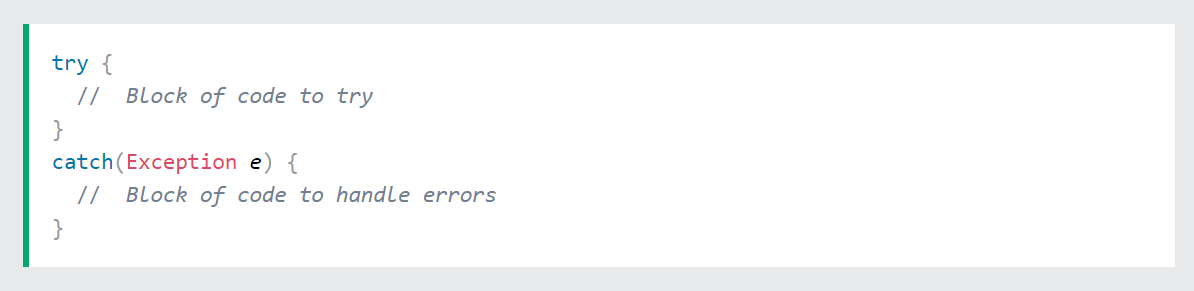
### Java try and catch

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.

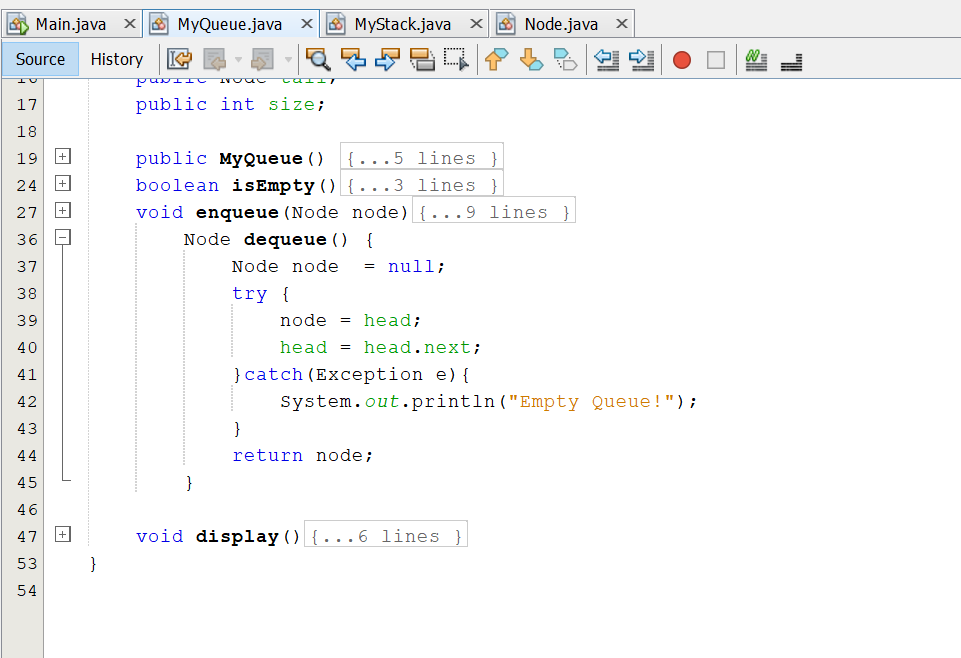
The try and catch keywords come in pairs:

Syntax

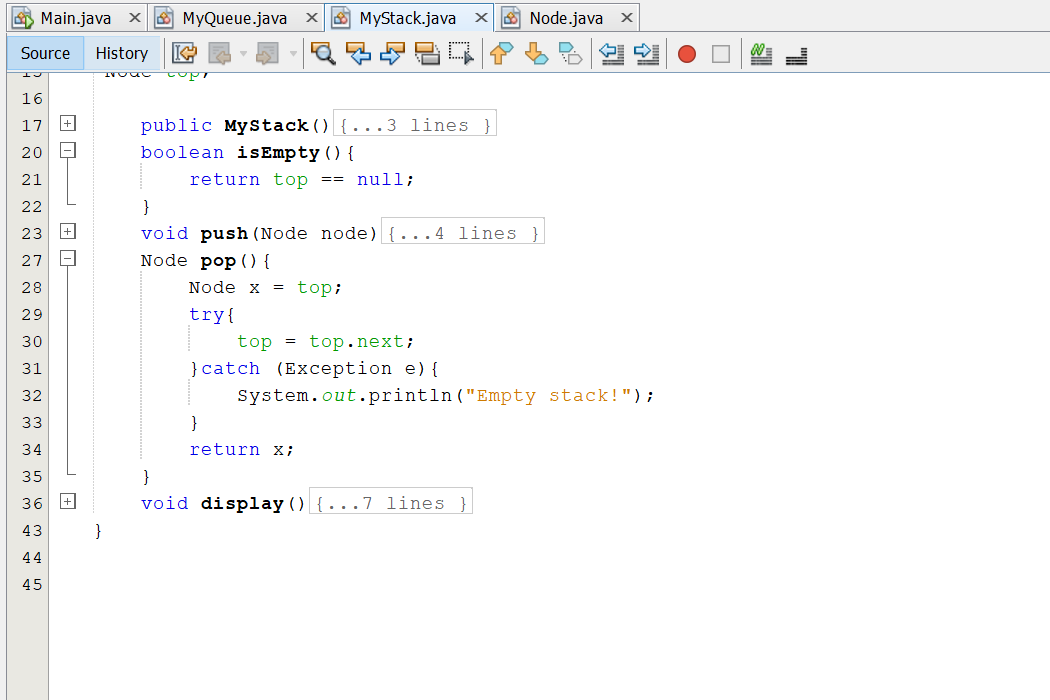


When the user presses the keys to select the function, the user only allowed to select numbers, if you press a letter, the program will stop and notcan keep running If an error occurs, I can use try...catch to catch the error and execute some code to handle it

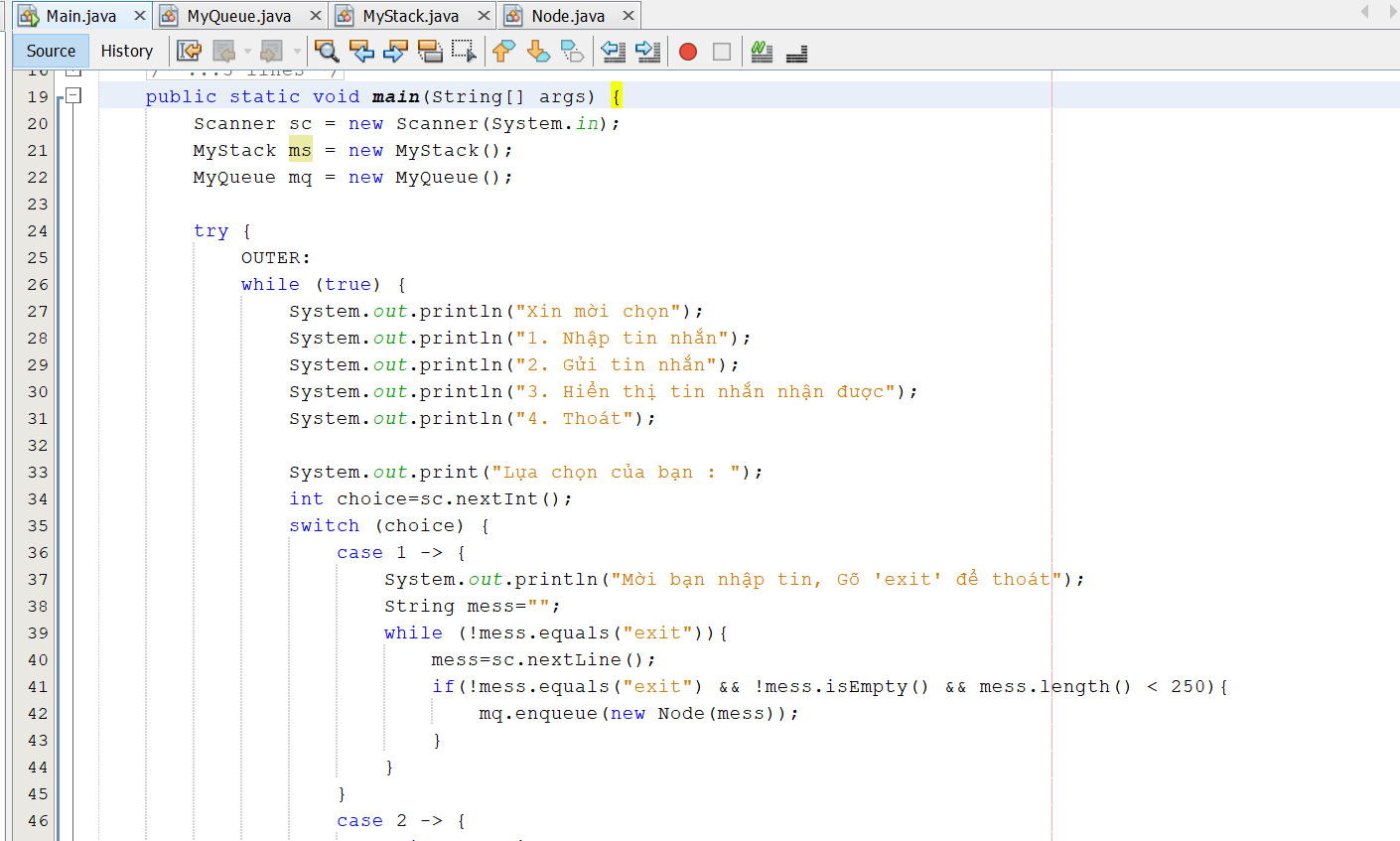
for class MyQueue :



Class MyStack :



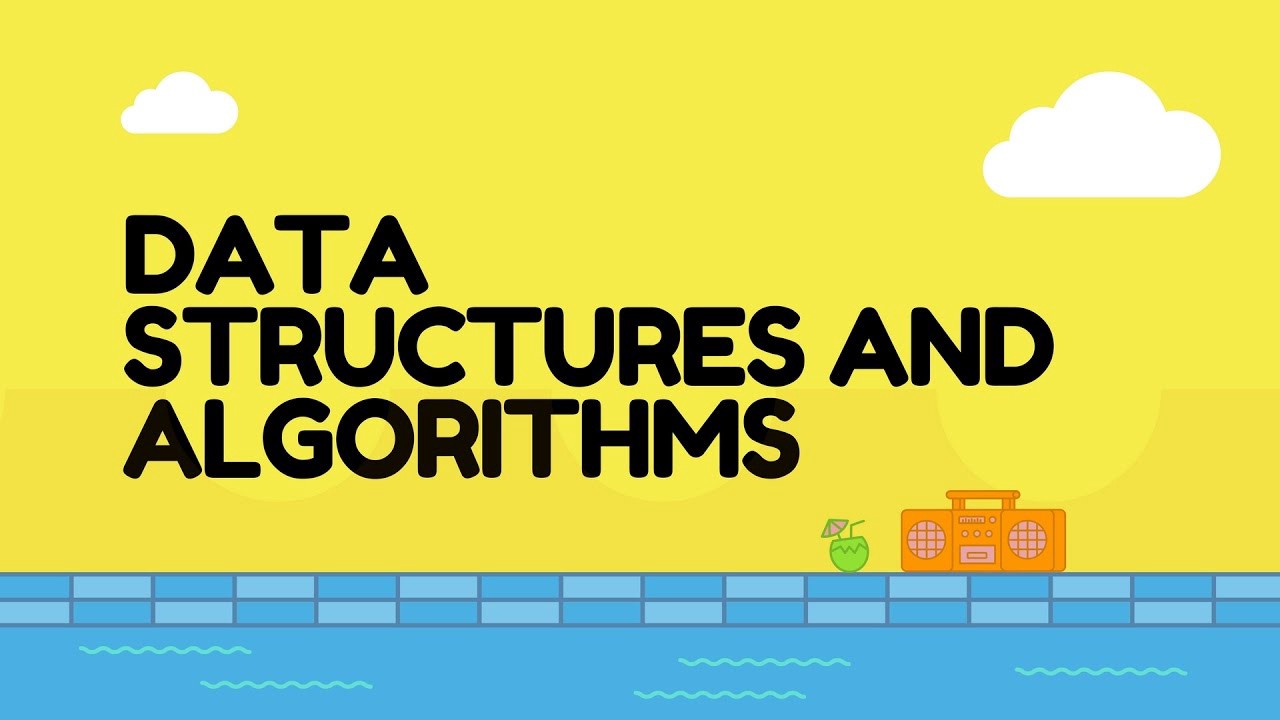
Class Main:



## **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.



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

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

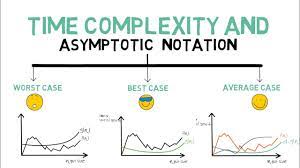
Sorting Algorithm: Algorithm to sort the elements in a certain order.

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.

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(n2). 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*

Ο Notation

Ω Notation

θ Notation

**Asymptotic Notations**

**Θ 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.

3n3 + 6n2 + 6000 = Θ(n3)

Dropping lower order terms is always fine because there will always be a number(n) after which Θ(n3) has higher values than Θ(n2) 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.

Lightbox

* 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 Ω**: Just as Big O notation provides an upper asymptote of a function, the notation Ω provides a lower asymptote.

BigOmega

Ω 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.

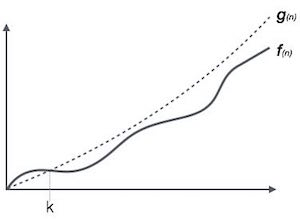
* For example:
  + f (n) =8+2n-3 ≥ 8-3

=7+(-3) ≥ 7 (g(n))

Thus, =7

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

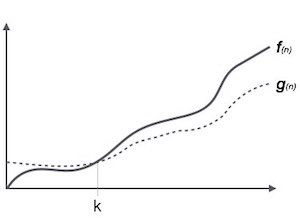
Ο(n) is a way to represent the upper asymptote of the running time of an algorithm. It estimates the worst case time complexity or the longest amount of time required by an algorithm (execution from start to finish). The graph shows the following:



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

Ο(f(n)) = { g(n) : if there exist c > 0 and n0 such that g(n) ≤ c.f(n) for all n > n0. }

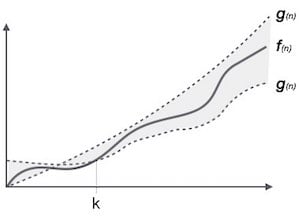
### 3.4 Omega Notation, in Data Structures and Algorithms

The Ω(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):

Ω(f(n)) ≥ { g(n) : if there exist c > 0 and n0 such that g(n) ≤ c.f(n) for all n > n0. }

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

The θ(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:

Some popular Asymptotic Notations in data structures and algorithms

constant – Ο(1)

logarithm – (log n)

Linear (Linear) – Ο(n)

n log n – Ο(n log n)

Quadratic – Ο(n2)

Degree 3 (cubic) – (n3)

Polynomial – nΟ(1)

Exponential – 2Οn

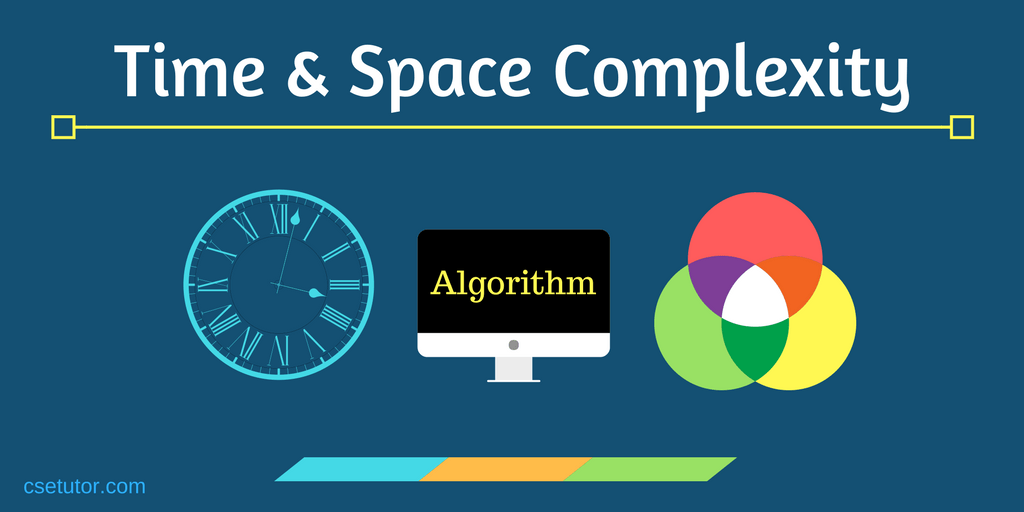
# **IV. Determine two ways in which the efficiency of an algorithm can be measured, illustrating your answer with an example.**

* 1. What is space complexity?

1. 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 if often expressed asymptotically in big O notation, such as O(n), O(nlog(n)) … where **n** is the input size in units of bits needed to represent the input



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(n2) space.

# **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. 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].