"Number Sorter"

"Handling of data and basic sorting algorithms."

"The presentation was made by" José Manuel Alonso Tirado

Numerical Order

Project Context:
In the realm of programming, the ability to sort data is fundamental. Our development team has set out to create a Java program that allows users to sort a list of numbers entered by themselves. This project will provide valuable practice for students in handling data and basic sorting algorithms.

Project Description:
The aim of this project is to develop a Java program that enables users to input a list of numbers and sort it in ascending or descending order, according to their choice. The program should be able to handle different quantities of numbers and ensure an efficient and accurate sorting process.

Project Objectives:
1. Understand the basic concepts of data input and control structures in Java.
2. Implement simple sorting algorithms, such as the bubble sort method, the selection sort method, or your preferred method, to sort a list of numbers.
3. Develop a simple user interface that allows users to input a list of numbers and select the desired type of sorting.
4. Perform thorough testing to verify the accuracy and efficiency of the sorting process.

Bonus:
The company requesting this program is from China. The number 4 is considered unlucky in their culture because the word "four" - both in Mandarin and Cantonese - sounds very similar to "death." For this reason, this number cannot be entered into the list.

Analyze the problem

Briefly elaborate on what you want to discuss.

Sorting numbers is a fundamental operation in computing and plays a crucial role in various applications, ranging from organizing data for efficient retrieval to facilitating efficient search algorithms.

In programming, sorting algorithms are essential tools,these algorithms come in various forms, each with its own set of advantages and disadvantages in terms of efficiency and implementation complexity.

In this documentation, we will delve into the realm of classification algorithms, exploring their importance, common types, and the underlying principles behind their functionality. We will also discuss the implementation of sorting algorithms in Java, focusing on the algorithmic logic and the classes involved in the process.

Algorithm and Efficiency

An algorithm is a set of well-defined instructions or rules that must be followed in a specific sequence to solve a problem or perform a task. In the context of sorting, algorithms are used to arrange elements in a specified order, such as ascending or descending.

Sorting algorithms play a crucial role in various applications, including database management, web search engines, operating systems, e-commerce platforms, and social media platforms. They enable efficient organization, retrieval, and presentation of data, enhancing the performance and usability of systems across different domains.

Complexity and Notation

Algorithmic complexity refers to the efficiency of an algorithm in terms of its resource usage, such as time and space. It is often expressed using Big O notation, which provides an upper bound on the growth rate of the algorithm's time or space requirements as the input size increases.

Bubble Sort

Is indeed one of the simplest sorting algorithms to understand and implement. Its basic idea is to repeatedly step through the list, compare adjacent elements, and swap them if they are in the wrong order. This process continues until the list is sorted.

While Bubble Sort is straightforward and easy to implement, it is not the most efficient sorting algorithm, especially for large lists. Its time complexity is O(n^2) in the worst and average cases, making it less suitable for large datasets compared to more efficient algorithms like QuickSort or MergeSort.

Selection Sort:

Selection sort is an in-place comparison sorting algorithm that divides the input list into two parts: the sorted sublist and the unsorted sublist.
It repeatedly finds the minimum element from the unsorted sublist and swaps it with the first unsorted element.

Despite its simplicity, selection sort is also inefficient on large lists, with a time complexity of O(n^2).

- *Analogy: Organizing Books on a Shelf*
  - *Imagine you have a shelf full of books in random order, and you want to arrange them in ascending order by their heights.*
  - *Selection sort works similarly to how you might organize these books:*
    i. *You start at one end of the shelf and look for the smallest book.*
    ii. *Once you find it, you swap it with the book at the first position.*
    iii. *Then, you move to the next position and repeat the process, finding the smallest book from the remaining unsorted books and swapping it into place.*
    iv. *You continue this process until all the books are sorted.*

Insertion Sort:

Insertion sort is a simple sorting algorithm that builds the final sorted list one element at a time.
It takes each element from the input list and inserts it into its correct position in the already-sorted part of the list.
While insertion sort is efficient for small lists or nearly sorted lists, it has an average and worst-case time complexity of O(n^2).

- *Analogy: Sorting Playing Cards in Your Hand*
  - *Imagine you are holding a hand of playing cards, and they are not in any particular order. You want to arrange them in ascending order by their values.*
  - *Insertion sort works similarly to how you might sort these cards:*
    i. *You start with one card in your hand, considering it as the sorted part.*
    ii. *Then, you pick up one card at a time from the unsorted part of your hand and insert it into the correct position among the cards already sorted in your hand.*
    iii. *You repeat this process until all cards are sorted in your hand.*

Merge Sort:

Merge sort is a divide-and-conquer sorting algorithm that divides the input list into smaller sublists, sorts each sublist, and then merges them back together.
It's known for its stable sorting behavior and guarantees O(n log n) time complexity in all cases, making it efficient for large lists.
Merge sort requires additional space proportional to the size of the input list due to its divide-and-conquer approach.

- *Analogy: Merging Sorted Lists*
  - *Imagine you have two piles of sorted papers, and you want to combine them into one sorted pile.*
  - *Merge sort works similarly to how you might merge these sorted piles:*

    i. *You start with the two sorted piles and take the top paper from each pile.*
    ii. *Compare the papers, and put the smaller one on the bottom of the new pile.*
    iii. *Repeat this process, always taking the smaller of the two top papers until one of the piles is empty.*
    iv. *Then, you simply place the remaining papers from the non-empty pile onto the bottom of the new pile, as they are already sorted.*
    v. *The new pile is now a sorted combination of the two original piles.*

But what is the complexity of an algorithm?

The time complexity of an algorithm, often denoted as its "complexity," describes how long an algorithm takes to execute based on the size of the input. In the case of Bubble Sort, its time complexity is expressed as O(n^2), where "n" represents the size of the data set to be sorted.

For example, in Insertion Sort:

- Best Case: O(n) (when the list is already sorted)
- Worst Case: O(n^2) (when the list is sorted in reverse order)
- Average Case: O(n^2)

QuickSort:

- Best Case: O(n log n)
- Worst Case: O(n^2) (when poorly chosen pivots lead to unbalanced partitions)
- Average Case: O(n log n)

MergeSort:

- Best Case: O(n log n)
- Worst Case: O(n log n)
- Average Case: O(n log n)

Recommended Use:

Insertion Sort:
- Suitable for small lists (up to a few hundred elements) or nearly sorted lists.
- Simple to implement and efficient for small or nearly sorted lists.

QuickSort:
- Ideal for large lists and cases where a good pivot selection strategy can be employed.
- Generally very fast in practice and requires little additional memory (in-place).

MergeSort:
- Recommended for very large lists and cases where a stable sorting algorithm is needed.
- Guarantees O(n log n) performance in all cases and maintains stability.

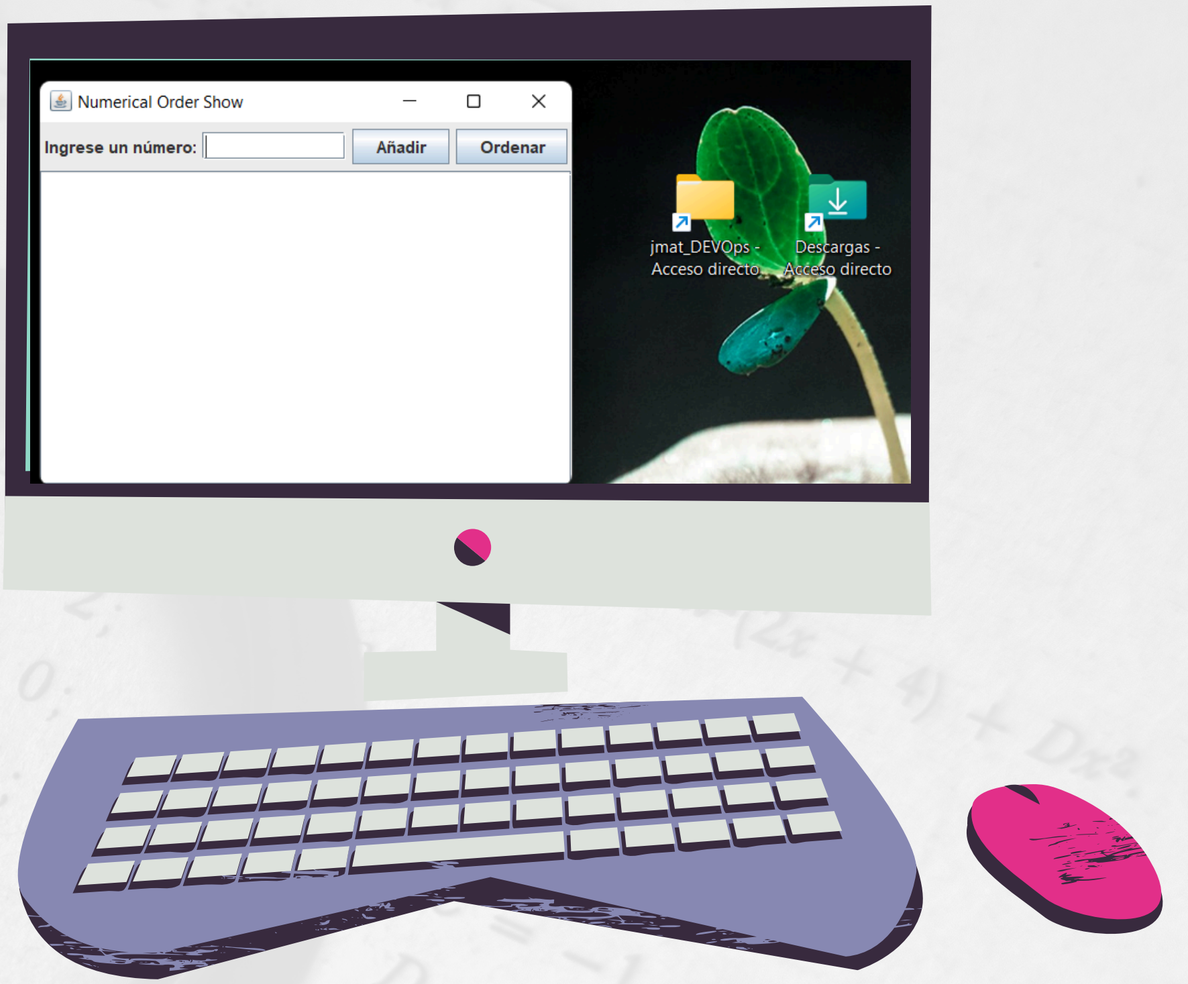
Classification by Array Length

For small lists (up to a few hundred elements), Insertion Sort is generally the best choice due to its simplicity and good performance.

For medium-sized lists (several hundred to a few thousand elements), QuickSort is often the preferred option due to its practical efficiency, especially with effective pivot selection strategies.

For large lists (thousands to millions of elements), MergeSort is a reliable choice due to its guaranteed O(n log n) complexity in all cases and stability. QuickSort can also be efficient but requires careful pivot selection to avoid worst-case scenarios

Write mine principal topic or idea



The main idea is to build a window that will have a form to collect the numbers and two buttons, one to accept the entered number and the other to perform the reorder option. The application can continue accepting new numbers and will reorder them.

The window hava a title Numerical Order Show.

The chosen algorithm is the bubble algorithm due to its simplicity, although with a high computational cost, it does not affect this small example. Another choice could have been the insertion.
this algorithm is Suitable for small lists (up to a few hundred items) or nearly ordered lists.

"Let's get to work"

Code view

```java
public class NumericalOrderGui extends JFrame {
    private final JTextField numberInput;
    private final JTextArea numberListArea;
    private final DefaultListModel<Integer> numberListModel;

    public NumericalOrderGui() {
        setTitle(title:"數字順序顯示---Numerical Order Show");
        setSize(width:700, height:400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(c:null);
        setLayout(new BorderLayout());

        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new FlowLayout());

        JLabel numberLabel = new JLabel(text:"插入一個數字 / Insert a number:
        numberInput = new JTextField(columns:10);
        JButton addButton = new JButton(text:"添加 / Add");
        JButton sortButton = new JButton(text:"命令 / Order");

        inputPanel.add(numberLabel);
        inputPanel.add(numberInput);
        inputPanel.add(addButton);
        inputPanel.add(sortButton);

        numberListModel = new DefaultListModel<>();
        JList<Integer> numberList = new JList<>(numberListModel);
        numberListArea = new JTextArea();
        numberListArea.setEditable(b:false);

        add(new JScrollPane(numberList), BorderLayout.CENTER);
        add(inputPanel, BorderLayout.NORTH);

        addButton.addActionListener((ActionEvent e) -> {
            addNumber();
        });

        sortButton.addActionListener((ActionEvent e) -> {
            sortNumbers();
        });
    }
```

NumericalOrder.java | NumericalOrderGui.java

src > main > java > jmat_f5bosco > NumericalOrder.java > Language Support for Java(TM) by Red Hat > NumericalOrder > bubbleSort(int[])

```java
package jmat_f5bosco;

public class NumericalOrder {
    public static void bubbleSort(int[] array) {
        int aux;
        boolean cambio;

        do {
            cambio = false;
            for (int i = 1; i < array.length; i++) {
                if (array[i] < array[i - 1]) {
                    // Intercambiar los elementos
                    aux = array[i];
                    array[i] = array[i - 1];
                    array[i - 1] = aux;
                    cambio = true;
                }
            }
        } while (cambio);
    }
}
```

Imports: We import the necessary classes from the java.awt and javax.swing packages for creating the graphical user interface.

Class Declaration of NumericalOrderGui:

We declare a class called NumericalOrderGui that extends the JFrame class, meaning it is a window in the graphical user interface.
This class will be responsible for displaying the user interface for sorting numbers.

```
public class NumericalOrderGui extends JFrame {
    private final JTextField numberInput;
    private final JTextArea numberListArea;
    private final DefaultListModel<Integer> numberListModel;
```

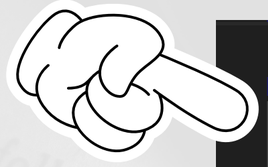
Campos de la clase:

numberInput: A field representing a text input field where the user can enter numbers (Chinese/English).
numberListArea: A text area where the entered numbers will be displayed.
numberListModel: A list model that contains the numbers entered by the user.

The first section of the code defines the basic structure of the window that will be displayed to the user.

Constructor:

We create a constructor for the NumericalOrderGui class.
Within the constructor, we set up the properties of the window, such as the title, size, close operation, position, and the layout of the interface.



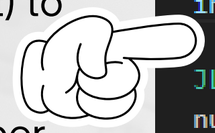
```
public NumericalOrderGui() {
    setTitle(title:"數字順序顯示---Numerical Order Show");
    setSize(width:700, height:400);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLocationRelativeTo(c:null);
    setLayout(new BorderLayout());
```

We create an input panel (inputPanel) with a flow layout (FlowLayout) to organize the elements horizontally in a line.

We create a label (numberLabel) to prompt the user to enter a number.

We create a text field (numberInput) where the user can enter numbers.

We create two buttons (addButton and sortButton) to add numbers to the list and sort the entered numbers, respectively.



```
JPanel inputPanel = new JPanel();
inputPanel.setLayout(new FlowLayout());

JLabel numberLabel = new JLabel(text:"插入一個數字 / Insert a number:");
numberInput = new JTextField(columns:10);
JButton addButton = new JButton(text:"添加 / Add");
JButton sortButton = new JButton(text:"命令 / Order");

inputPanel.add(numberLabel);
inputPanel.add(numberInput);
inputPanel.add(addButton);
inputPanel.add(sortButton);
```

- We add the components to the input panel (inputPanel).
- We create a list model (numberListModel) to hold the numbers entered by the user.
- We create a list (numberList) that will use the previously created list model to display the entered numbers.

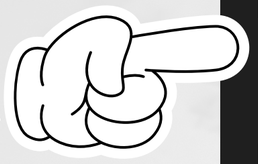
- We create a text area (numberListArea) that will be used to display the entered numbers.
- We add a scroll pane (JScrollPane) that will contain the list of numbers and place it in the center of the window.
- We add the input panel (inputPanel) at the top of the window.

```
numberListModel = new DefaultListModel<>();
JList<Integer> numberList = new JList<>(numberListModel);
numberListArea = new JTextArea();
numberListArea.setEditable(b:false);

add(new JScrollPane(numberList), BorderLayout.CENTER);
add(inputPanel, BorderLayout.NORTH);
```

We associate an ActionListener with the buttons (addButton and sortButton) to handle click events on the buttons.

When the "Add" button is clicked, the addNumber() method is called.

When the "Sort" button is clicked, the sortNumbers() method is called.



```java
addButton.addActionListener((ActionEvent e) -> {
    addNumber();
});

sortButton.addActionListener((ActionEvent e) -> {
    sortNumbers();
});
```

Visual Studio Code itself refactors the code for me and suggests lambda expressions??? I have to check what they are ;)

Method addNumber():

This method handles the logic for adding numbers to the list. It attempts to convert the text entered in the text field (numberInput) to an integer. If the number 4 is entered, it displays an error message indicating that the number 4 is considered unlucky in Chinese culture and cannot be added to the list. If a valid number is entered, it adds it to the list model (numberListModel) and clears the text field (numberInput).

```java
private void addNumber() {
    try {
        int number = Integer.parseInt(numberInput.getText());
        if (number == 4) {
            JOptionPane.showMessageDialog(this, message:"數字4在中國文化中被認為是不吉利的，不能進入該列表 / The number 4 is considered unlucky in Chinese culture and cannot be entered into the list.", title:"Error",
            JOptionPane.ERROR_MESSAGE);
            return;
        }
        numberListModel.addElement(number);
        numberInput.setText(t:"");
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, message:"Por favor, ingrese un número entero válido.", title:"Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

```java
private void sortNumbers() {
    int size = numberListModel.getSize();
    int[] array = new int[size];
    for (int i = 0; i < size; i++) {
        array[i] = numberListModel.getElementAt(i);
    }

    NumericalOrder.bubbleSort(array);

    numberListModel.clear();
    for (int num : array) {
        numberListModel.addElement(num);
    }
}
```

Method sortNumbers():

This method handles the logic for sorting the entered numbers.
It gets the size of the list of numbers.
It creates an integer array and copies the elements from the list to the array.
It uses the bubbleSort() method from the NumericalOrder class to sort the array.
It clears the list model and adds the sorted numbers to the list model.

Method main():

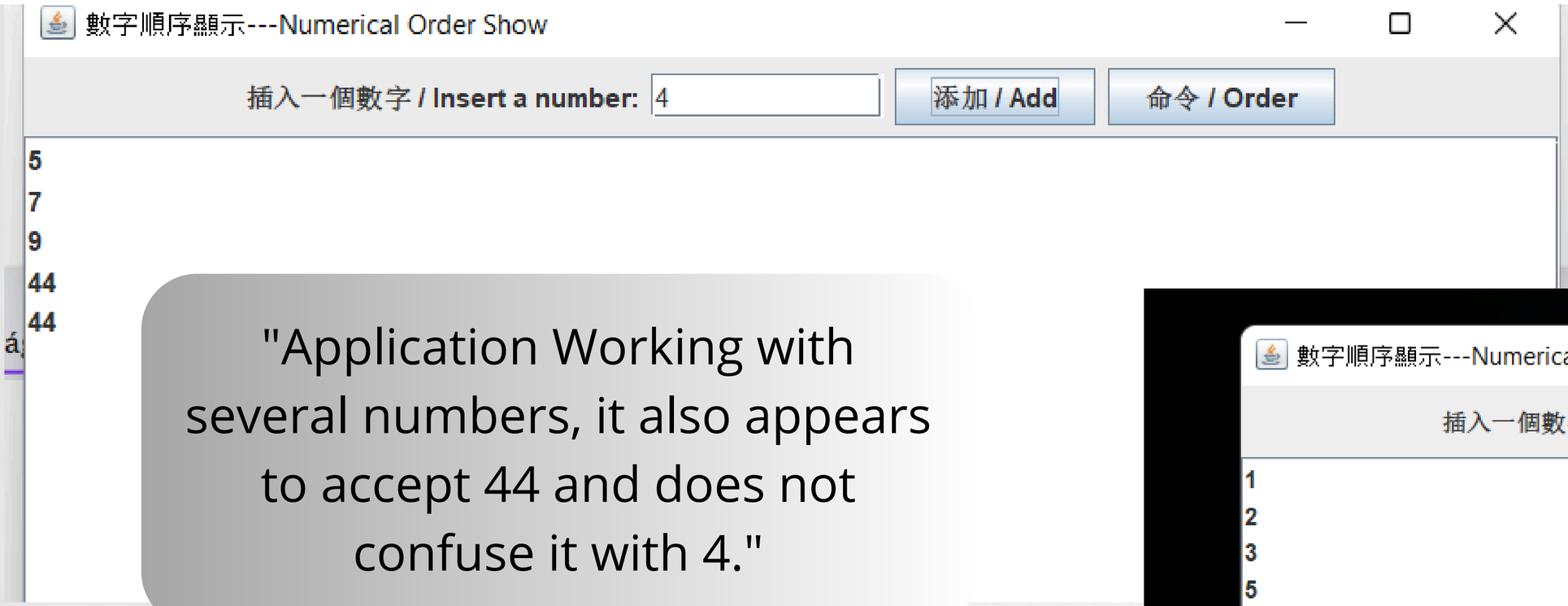
The main() method starts the application.
It uses SwingUtilities.invokeLater() to ensure that the creation of the graphical user interface is performed on the Swing event dispatch thread to avoid concurrency issues.
It creates an instance of the NumericalOrderGui class and makes it visible.

```java
Run main | Debug main | Run | Debug
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        new NumericalOrderGui().setVisible(b:true);
    });
}
}
```

Another class in your project contains the bubbleSort method. Here is a typical implementation of a bubble sort algorithm:

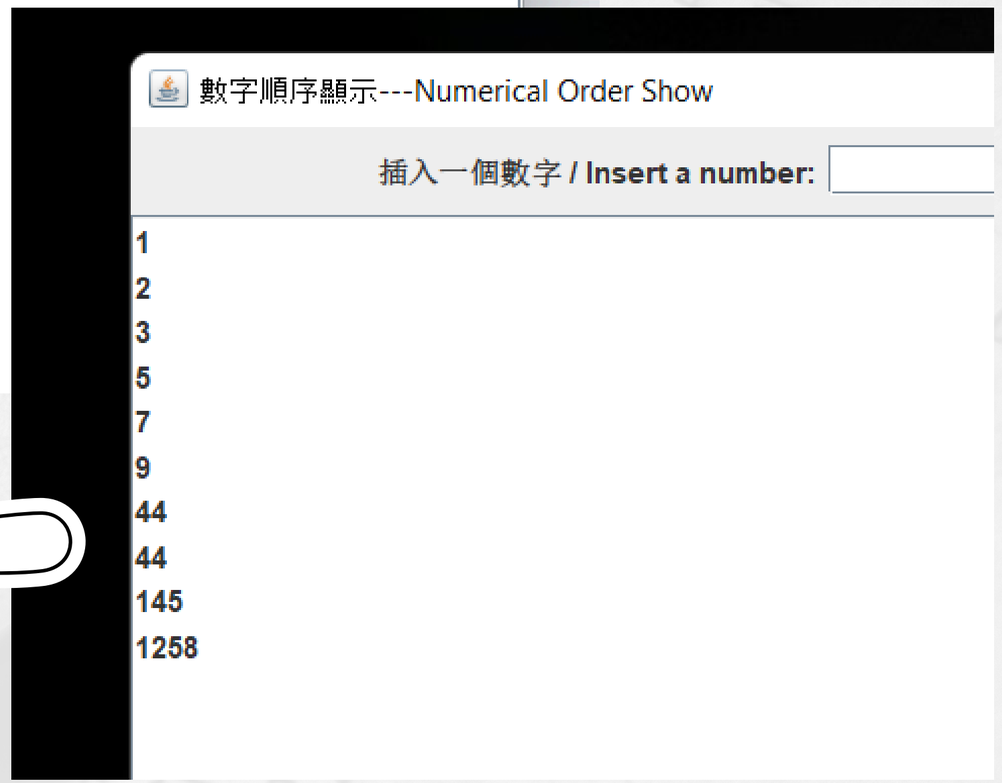
src > main > java > jmat_f5bosco > NumericalOrder.java > ...

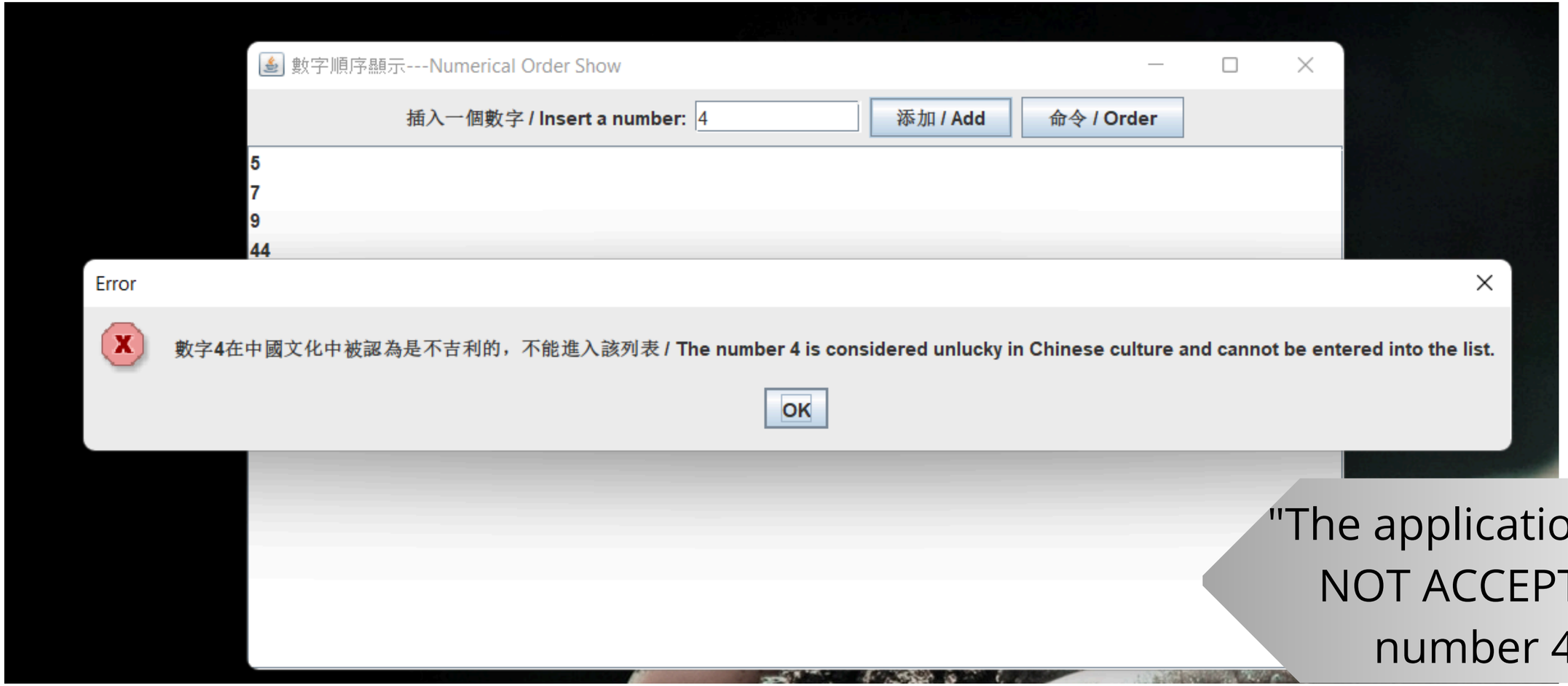
```java
package jmat_f5bosco;

public class NumericalOrder {
    public static void bubbleSort(int[] array) {
        int aux;
        boolean cambio;

        do {
            cambio = false;
            for (int i = 1; i < array.length; i++) {
                if (array[i] < array[i - 1]) {
                    // Intercambiar los elementos
                    aux = array[i];
                    array[i] = array[i - 1];
                    array[i - 1] = aux;
                    cambio = true;
                }
            }
        } while (cambio);
    }
}
```



"Application Working with several numbers, it also appears to accept 44 and does not confuse it with 4."

"Application with several numbers already sorted."





"The application does NOT ACCEPT the number 4."

Documentation

BIBLIOGRAPHY

https://docs.oracle.com/javase/8/docs/api/javax/swing/JPanel.html

http://www.edu4java.com/es/java/joptionpane-showmessagedialog-showinputdialog.html