16.b)Write a Java program to find maximum of n numbers.

```
import java.util.Scanner;

public class MaxNumberFinder {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the number of elements
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();

        // Prompt the user to enter the numbers
        System.out.println("Enter " + n + " numbers:");

        // Initialize maxNumber with the minimum possible value for integers
        int maxNumber = Integer.MIN_VALUE;

        // Loop to input and find the maximum number
        for (int i = 0; i < n; i++) {
            System.out.print("Enter number " + (i + 1) + ": ");
            int currentNumber = scanner.nextInt();

            // Update maxNumber if the current number is greater
            if (currentNumber > maxNumber) {
                maxNumber = currentNumber;
            }
        }

        // Display the maximum number
        System.out.println("The maximum number is: " + maxNumber);

        scanner.close();
    }
}
```

Algorithm:
Import Statement:

```
import java.util.Scanner;
```

This line imports the Scanner class from the java.util package, which is used to take input from the user.

Main Method:

```
public static void main(String[] args) {
```

This line declares the main method, which is the entry point of the program.

Scanner Initialization:

```
Scanner scanner = new Scanner(System.in);
```

Here, a Scanner object named scanner is created to read input from the user.

Enter the Number of Elements:

```
System.out.print("Enter the number of elements: ");
int n = scanner.nextInt();
```

These lines prompt the user to enter the number of elements (n) that they want to compare.

Enter Numbers and Find the Maximum:

```
System.out.println("Enter " + n + " numbers:");
int maxNumber = Integer.MIN_VALUE;
```

These lines initialize the maxNumber variable with the minimum possible value for integers. The program then enters a loop to input n numbers and find the maximum.

```
j
for (int i = 0; i < n; i++) {
    System.out.print("Enter number " + (i + 1) + ": ");
    int currentNumber = scanner.nextInt();

    if (currentNumber > maxNumber) {
        maxNumber = currentNumber;
    }
}
```

Inside the loop, the program prompts the user to enter each number (currentNumber). It compares each entered number with the current maximum (maxNumber). If the entered number is greater than the current maximum, it updates maxNumber with the entered number.

Display the Maximum Number:

```
System.out.println("The maximum number is: " + maxNumber);
```

After the loop, the program prints the maximum number found.

Close Scanner:

```
scanner.close();
```

Finally, the scanner object is closed to release resources associated with it.

This program efficiently finds the maximum of n numbers entered by the user using a loop and a comparison mechanism.

OUTPUT:

```
Enter the number of elements: 5
Enter 5 numbers:
Enter number 1: 1
Enter number 2: 2
Enter number 3: 3
```

```
Enter number 4: 4
Enter number 5: 5
The maximum number is: 5
================================================================================
======================
```

17.a)) Write a Java program to implement User interface screen for Online shopping using AWF controls.

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class OnlineShoppingUI extends Frame implements ActionListener {
    private TextField itemNameField, quantityField, priceField;
    private TextArea displayArea;

    public OnlineShoppingUI() {
        // Set layout manager
        setLayout(new FlowLayout());

        // Create and add components
        Label itemNameLabel = new Label("Item Name:");
        itemNameField = new TextField(20);

        Label quantityLabel = new Label("Quantity:");
        quantityField = new TextField(5);

        Label priceLabel = new Label("Price:");
        priceField = new TextField(10);

        Button addButton = new Button("Add to Cart");
        addButton.addActionListener(this);

        displayArea = new TextArea(10, 40);

        add(itemNameLabel);
        add(itemNameField);
        add(quantityLabel);
        add(quantityField);
        add(priceLabel);
        add(priceField);
        add(addButton);
        add(displayArea);
```

```
        // Set frame properties
        setTitle("Online Shopping");
        setSize(400, 300);
        setVisible(true);

        // Handle window close event
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent windowEvent) {
                System.exit(0);
            }
        });
    }

    public static void main(String[] args) {
        new OnlineShoppingUI();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Add to Cart")) {
            String itemName = itemNameField.getText();
            int quantity = Integer.parseInt(quantityField.getText());
            double price = Double.parseDouble(priceField.getText());

            double total = quantity * price;

            displayArea.append("Item: " + itemName + "\n");
            displayArea.append("Quantity: " + quantity + "\n");
            displayArea.append("Price: $" + price + "\n");
            displayArea.append("Total: $" + total + "\n");
            displayArea.append("----------------------\n");

            // Clear input fields
            itemNameField.setText("");
            quantityField.setText("");
            priceField.setText("");
        }
    }
}
```

Algorithm:
Certainly! Here's a high-level algorithm for the Online Shopping User Interface program:

1. **Create the OnlineShoppingUI Class:**
   - Declare the class `OnlineShoppingUI` that extends `Frame`.
   - Define instance variables for `TextField` (itemNameField, quantityField, priceField), `TextArea` (displayArea), and other necessary components.

2. **Initialize the GUI Components:**
   - Set the layout manager for the frame.
   - Create labels, text fields, a button, and a text area for displaying items.
   - Add these components to the frame.

3. **Handle GUI Events:**
   - Implement the `ActionListener` interface to handle button click events.
   - Override the `actionPerformed` method to process the "Add to Cart" button click.
   - Retrieve values from text fields (item name, quantity, price).
   - Calculate the total price and display the item details in the text area.
   - Clear the input fields after adding an item to the cart.

4. **Set Up the Window:**
   - Set the title of the frame to "Online Shopping."
   - Set the size of the frame.
   - Make the frame visible.

5. **Handle Window Close Event:**
   - Add a `WindowAdapter` to handle the window close event.
   - Override the `windowClosing` method to exit the program when the user closes the window.

6. **Create the Main Method:**
   - Implement the `main` method.
   - Instantiate an object of the `OnlineShoppingUI` class to create and display the GUI.

7. **Run the Program:**
   - Compile and run the program to launch the Online Shopping UI.
   - Interact with the UI by entering item details and clicking the "Add to Cart" button.
   - Close the window to exit the program.

==========
17.b)Write a Java program to print Fibonacci series

```
import java.util.Scanner;

public class FibonacciSeries {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of terms in the Fibonacci series: ");
        int n = scanner.nextInt();

        System.out.println("Fibonacci Series:");
        printFibonacciSeries(n);
```

```
        scanner.close();
    }

    static void printFibonacciSeries(int n) {
        int firstTerm = 0, secondTerm = 1;

        for (int i = 0; i < n; i++) {
            System.out.print(firstTerm + " ");

            int nextTerm = firstTerm + secondTerm;
            firstTerm = secondTerm;
            secondTerm = nextTerm;
        }
    }
}
```

Algorithm:

Certainly! Below is a high-level algorithm for generating and printing the Fibonacci series in Java:

1. **Start:**
2. Initialize variables `firstTerm` and `secondTerm` to 0 and 1, respectively.
3. Read the number of terms `n` from the user.
4. Print "Fibonacci Series:" to the console.
5. **Loop (for i from 0 to n-1):**
   1. Print the value of `firstTerm`.
   2. Calculate the next term as the sum of `firstTerm` and `secondTerm`.
   3. Update `firstTerm` with the value of `secondTerm`.
   4. Update `secondTerm` with the value of the next term.
6. **End Loop**
7. **End**

OUTPUT:

```
Enter the number of terms in the Fibonacci series: 20
Fibonacci Series:
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
```

==========

18. Write a java program to create Tabbed Pane with Tree and Table as tab and to demonstrate JTree and JTable components with JscrollPane.

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.tree.DefaultMutableTreeNode;
```

```
import javax.swing.tree.DefaultTreeModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class EmployeeManagementApp {
    private static DefaultTableModel employeeTableModel;

    public static void main(String[] args) {
        JFrame frame = new JFrame("Employee Management Application");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(600, 400);

        JTabbedPane tabbedPane = new JTabbedPane();

        JPanel employeePanel = createEmployeeTab();
        JPanel departmentPanel = createDepartmentTab();

        tabbedPane.addTab("Employee Information", employeePanel);
        tabbedPane.addTab("Departments", departmentPanel);

        frame.add(tabbedPane);
        frame.setVisible(true);
    }

    private static JPanel createEmployeeTab() {
        JPanel employeePanel = new JPanel(new BorderLayout());

        String[] columnNames = {"ID", "Name", "Position"};
        employeeTableModel = new DefaultTableModel(columnNames, 0);

        JTable employeeTable = new JTable(employeeTableModel);
        JScrollPane scrollPane = new JScrollPane(employeeTable);

        JButton addButton = new JButton("Add Employee");
        addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String id = JOptionPane.showInputDialog(null, "Enter Employee ID:");
                String name = JOptionPane.showInputDialog(null, "Enter Employee
Name:");
                String position = JOptionPane.showInputDialog(null, "Enter Employee
Position:");

                employeeTableModel.addRow(new Object[]{id, name, position});
            }
        });

        employeePanel.add(scrollPane, BorderLayout.CENTER);
        employeePanel.add(addButton, BorderLayout.SOUTH);
```

```
        return employeePanel;
    }

    private static JPanel createDepartmentTab() {
        JPanel departmentPanel = new JPanel(new BorderLayout());

        DefaultMutableTreeNode root = new DefaultMutableTreeNode("Departments");
        DefaultMutableTreeNode department1 = new DefaultMutableTreeNode("HR");
        DefaultMutableTreeNode department2 = new
DefaultMutableTreeNode("Engineering");

        root.add(department1);
        root.add(department2);

        JTree departmentTree = new JTree(root);
        JScrollPane scrollPane = new JScrollPane(departmentTree);

        JButton addButton = new JButton("Add Department");
        addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String departmentName = JOptionPane.showInputDialog(null, "Enter
department name:");
                if (departmentName != null && !departmentName.isEmpty()) {
                    DefaultMutableTreeNode newDepartment = new
DefaultMutableTreeNode(departmentName);
                    root.add(newDepartment);
                    ((DefaultTreeModel) departmentTree.getModel()).reload(root);
                }
            }
        });

        departmentPanel.add(scrollPane, BorderLayout.CENTER);
        departmentPanel.add(addButton, BorderLayout.SOUTH);

        return departmentPanel;
    }
}
```

Algorithm:

Certainly! Below is a high-level algorithm for the Employee Management Application program:

1. **Start:**
2. Create a `JFrame` named `frame`.
3. Set the default close operation of the frame to `EXIT_ON_CLOSE`.
4. Set the size of the frame to 600 x 400 pixels.

5. Create a `JTabbedPane` named `tabbedPane`.
6. Create a `JPanel` named `employeePanel` using the `createEmployeeTab` method.
7. Create a `JPanel` named `departmentPanel` using the `createDepartmentTab` method.
8. Add tabs to the `tabbedPane`:
   - Add a tab named "Employee Information" with `employeePanel`.
   - Add a tab named "Departments" with `departmentPanel`.
9. Add the `tabbedPane` to the `frame`.
10. Make the frame visible.

createEmployeeTab:
1. Create a `JPanel` named `employeePanel` with a `BorderLayout`.
2. Create a `String` array named `columnNames` with values "ID", "Name", "Position".
3. Create a `DefaultTableModel` named `employeeTableModel` with `columnNames` and 0 rows.
4. Create a `JTable` named `employeeTable` with `employeeTableModel`.
5. Create a `JScrollPane` named `scrollPane` with `employeeTable`.
6. Create a `JButton` named `addButton` with the label "Add Employee" and an `ActionListener`.
   - Inside the `actionPerformed` method, use `JOptionPane` to input Employee ID, Name, and Position.
   - Add a new row with the input values to `employeeTableModel`.
7. Add `scrollPane` and `addButton` to `employeePanel`.
8. Return `employeePanel`.

createDepartmentTab:
1. Create a `JPanel` named `departmentPanel` with a `BorderLayout`.
2. Create a `DefaultMutableTreeNode` named `root` with the value "Departments".
3. Create `DefaultMutableTreeNode` nodes for "HR" and "Engineering" and add them to the `root`.
4. Create a `JTree` named `departmentTree` with the `root`.
5. Create a `JScrollPane` named `scrollPane` with `departmentTree`.
6. Create a `JButton` named `addButton` with the label "Add Department" and an `ActionListener`.
   - Inside the `actionPerformed` method, use `JOptionPane` to input the department name.
   - Create a new `DefaultMutableTreeNode` with the input name and add it to the `root`.
   - Reload the model to reflect changes in the tree.
7. Add `scrollPane` and `addButton` to `departmentPanel`.
8. Return `departmentPanel`.

End

This algorithm outlines the steps for creating the Employee Management Application. The application has two tabs: one for managing employee information and another for managing departments. The program uses `JTable` and `JTree` components to display and manage data, and it allows users to add new employees and departments dynamically.

==========

==========
18.b)(b) write a Java program to check whether given number is palindrome or not

```
import java.util.Scanner;

public class PalindromeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        if (isPalindrome(number)) {
            System.out.println(number + " is a palindrome.");
        } else {
            System.out.println(number + " is not a palindrome.");
        }

        scanner.close();
    }

    // Function to check if a number is a palindrome
    private static boolean isPalindrome(int number) {
        int originalNumber = number;
        int reversedNumber = 0;

        while (number > 0) {
            int digit = number % 10;
            reversedNumber = reversedNumber * 10 + digit;
            number /= 10;
        }

        return originalNumber == reversedNumber;
    }
}
```

Algorithm:

Certainly! Let's go through the provided Java program line by line:

```java
import java.util.Scanner;
```

This line imports the `Scanner` class from the `java.util` package, which is used to take input from the user.

```java
public class PalindromeChecker {
```

```

This line starts the definition of a class named `PalindromeChecker`. The class encapsulates the logic for checking whether a given number is a palindrome.

```java
    public static void main(String[] args) {
```

This line declares the main method, which is the entry point of the program. The program execution starts from the `main` method.

```java
        Scanner scanner = new Scanner(System.in);
```

Here, a `Scanner` object named `scanner` is created to read input from the user.

```java
        System.out.print("Enter a number: ");
```

This line prints the message "Enter a number: " to the console, prompting the user to input a number.

```java
        int number = scanner.nextInt();
```

This line reads an integer input from the user using the `nextInt` method of the `Scanner` class and assigns it to the variable `number`.

```java
        if (isPalindrome(number)) {
```

This line starts an `if` statement. It checks whether the result of calling the `isPalindrome` method with the input `number` is `true`.

```java
            System.out.println(number + " is a palindrome.");
```

If the condition in the `if` statement is true, this line prints a message to the console indicating that the entered number is a palindrome.

```java
        } else {
```
```

If the condition in the `if` statement is false, the program executes the code block inside the `else` statement.

```java
            System.out.println(number + " is not a palindrome.");
```

This line prints a message to the console indicating that the entered number is not a palindrome.

```java
        }

        scanner.close();
```

Finally, the `scanner` object is closed to release resources associated with it.

```java
    }

    // Function to check if a number is a palindrome
    private static boolean isPalindrome(int number) {
```

Here starts the definition of a private method named `isPalindrome`. This method takes an integer parameter (`number`) and returns a boolean value (`true` if the number is a palindrome, `false` otherwise).

```java
        int originalNumber = number;
        int reversedNumber = 0;
```

These lines declare two variables, `originalNumber` and `reversedNumber`, to store the original and reversed forms of the input number.

```java
        while (number > 0) {
            int digit = number % 10;
            reversedNumber = reversedNumber * 10 + digit;
            number /= 10;
        }
```

This `while` loop iterates through the digits of the original number, extracting the last digit (`digit`) in each iteration. It then builds the `reversedNumber` by multiplying it by 10 and adding the extracted digit. Finally, the last digit is removed from the `number`. This process continues until the `number` becomes 0.

```java
        return originalNumber == reversedNumber;
    }
}
```

This line returns `true` if the `originalNumber` is equal to the `reversedNumber`, indicating that the original number is a palindrome. Otherwise, it returns `false`. The class definition and the program end here.

OUTPUT:

```
Enter a number: 121
121 is a palindrome.
Enter a number: 1234
1234 is not a palindrome.
```

CHARACTER PALINDROME

```java
import java.util.Scanner;

public class PalindromeChecker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a word: ");
        String word = scanner.nextLine();

        if (isPalindrome(word)) {
            System.out.println(word + " is a palindrome.");
        } else {
            System.out.println(word + " is not a palindrome.");
        }

        scanner.close();
    }

    // Function to check if a word is a palindrome
    private static boolean isPalindrome(String word) {
        String reversedWord = "";

        // Reverse the word
        for (int i = word.length() - 1; i >= 0; i--) {
            reversedWord += word.charAt(i);
        }

        // Check if the original word is equal to the reversed word
        return word.equalsIgnoreCase(reversedWord);
    }
```

```
}
```

ALOGORITHM:

This program takes a word as input and checks whether it is a palindrome using the isPalindrome method. The isPalindrome method reverses the input word and compares it with the original word, ignoring case.

Here's how the program works:

It prompts the user to enter a word.
It calls the isPalindrome method to check if the entered word is a palindrome.
It prints the result to the console.
You can run this program and test it with different words to see if they are palindromes or not.
OUTPUT:

```
Enter a word: radar
radar is a palindrome.

Enter a word: san
san is not a palindrome.
```

==========

19.a)19. (a) Write a Java program to demonstrate different layouts(Flow Layout, Boarder Layout and GridBag Layout)

```
import java.awt.*;
import java.awt.event.*;

public class MovieTicketBookingApp {
    private Frame frame;
    private Panel customerInfoPanel, seatSelectionPanel;
    private Label nameLabel, ageLabel, movieLabel, showtimeLabel;
    private TextField nameField, ageField;
    private Choice movieChoice, showtimeChoice;
    private Button bookButton;
    private CheckboxGroup seatGroup;
    private Checkbox[] seatCheckboxes;

    public MovieTicketBookingApp() {
        frame = new Frame("Movie Ticket Booking");
        frame.setSize(400, 300);
        frame.setLayout(new BorderLayout());

        // Customer Info Panel with FlowLayout
        customerInfoPanel = new Panel(new FlowLayout(FlowLayout.LEFT));
```

```
        nameLabel = new Label("Name:");
        nameField = new TextField(20);
        ageLabel = new Label("Age:");
        ageField = new TextField(5);
        movieLabel = new Label("Movie:");
        movieChoice = new Choice();
        movieChoice.add("Avengers: Endgame");
        movieChoice.add("Spider-Man: No Way Home");
        movieChoice.add("The Matrix Resurrections");
        showtimeLabel = new Label("Showtime:");
        showtimeChoice = new Choice();
        showtimeChoice.add("10:00 AM");
        showtimeChoice.add("2:00 PM");
        showtimeChoice.add("7:00 PM");

        bookButton = new Button("Book Ticket");
        bookButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Handle ticket booking here
                String name = nameField.getText();
                String age = ageField.getText();
                String selectedMovie = movieChoice.getSelectedItem();
                String selectedShowtime = showtimeChoice.getSelectedItem();

                // Check if a seat is selected
                Checkbox selectedCheckbox = seatGroup.getSelectedCheckbox();
                String selectedSeat = (selectedCheckbox != null) ?
selectedCheckbox.getLabel() : "No seat selected";

                // Perform booking logic (not implemented in this example)
                System.out.println("Ticket booked for " + name + " (Age: " + age +
")");
                System.out.println("Movie: " + selectedMovie);
                System.out.println("Showtime: " + selectedShowtime);
                System.out.println("Seat: " + selectedSeat);
            }
        });

        customerInfoPanel.add(nameLabel);
        customerInfoPanel.add(nameField);
        customerInfoPanel.add(ageLabel);
        customerInfoPanel.add(ageField);
        customerInfoPanel.add(movieLabel);
        customerInfoPanel.add(movieChoice);
        customerInfoPanel.add(showtimeLabel);
        customerInfoPanel.add(showtimeChoice);
        customerInfoPanel.add(bookButton);

        // Seat Selection Panel with GridLayout
        seatSelectionPanel = new Panel(new GridLayout(4, 3));
```

```
        seatGroup = new CheckboxGroup();
        seatCheckboxes = new Checkbox[12];
        for (int i = 0; i < 12; i++) {
            seatCheckboxes[i] = new Checkbox("Seat " + (i + 1), seatGroup, false);
            seatSelectionPanel.add(seatCheckboxes[i]);
        }

        frame.add(customerInfoPanel, BorderLayout.NORTH);
        frame.add(seatSelectionPanel, BorderLayout.CENTER);

        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });

        frame.setVisible(true);
    }

    public static void main(String[] args) {
        new MovieTicketBookingApp();
    }
}
```

Alogriothm:

Below is a high-level algorithm for the provided Movie Ticket Booking application code:

1. Create a frame for the Movie Ticket Booking application.
2. Set the frame's size and layout to BorderLayout.

3. Create a "Customer Info" panel with FlowLayout:
   a. Add labels for "Name" and "Age."
   b. Add text fields for the customer's name and age.
   c. Add a label for "Movie" and a choice (dropdown) for selecting a movie.
   d. Add a label for "Showtime" and a choice (dropdown) for selecting a showtime.
   e. Add a "Book Ticket" button.
   f. Attach an ActionListener to the "Book Ticket" button to handle the booking process.

4. Create a "Seat Selection" panel with GridLayout:
   a. Create a CheckboxGroup for seat selection.
   b. Create checkboxes for each available seat (e.g., "Seat 1," "Seat 2," etc.) and add them to the panel.

5. Add the "Customer Info" panel to the frame's NORTH region.
6. Add the "Seat Selection" panel to the frame's CENTER region.

7. Implement the ActionListener for the "Book Ticket" button:

a. Retrieve the customer's name and age from the text fields.
b. Retrieve the selected movie and showtime from the choice components.
c. Retrieve the selected seat using the CheckboxGroup.
d. Perform any booking or reservation logic (not implemented in this example).
e. Print the booked ticket details, including the customer's name, age, selected movie, showtime, and seat.

8. Add a window listener to the frame to handle window closing events.

9. Set the frame's visibility to true, making it visible to the user.

10. The program will run, and users can enter their information, choose a movie, showtime, and seat. Upon clicking the "Book Ticket" button, the details of the booked ticket will be displayed.

This algorithm outlines the main steps of the provided Java code for the Movie Ticket Booking application. The actual code includes more detailed implementations of these steps.

OUTPUT:

```
Ticket booked for san (Age: 21)
Movie: Spider-Man: No Way Home
Showtime: 10:00 AM
Seat: Seat 4
```

==========

19.b) Write a Java program to sort the array elements in ascending order

```
public class ArraySorting {
    public static void main(String[] args) {
        int[] arrayToSort = {5, 2, 8, 1, 6, 3, 7, 4};

        System.out.println("Original Array: ");
        printArray(arrayToSort);

        // Sorting the array in ascending order using Bubble Sort
        bubbleSort(arrayToSort);

        System.out.println("Sorted Array (Ascending Order): ");
        printArray(arrayToSort);
    }

    // Bubble Sort algorithm for sorting in ascending order
    public static void bubbleSort(int[] array) {
        int n = array.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (array[j] > array[j + 1]) {
                    // Swap elements if they are in the wrong order
```

```
                    int temp = array[j];
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
    }

    // Utility method to print the elements of an array
    public static void printArray(int[] array) {
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

Algorithm:

Sure, let's break down the steps of the Bubble Sort algorithm:

Bubble Sort Algorithm:

1. **Start:**
   - Begin with the first element of the array.

2. **Compare Adjacent Elements:**
   - Compare the current element with the next element.
   - If the current element is greater than the next element, swap them.

3. **Move to the Next Pair:**
   - Move to the next pair of elements (from left to right).

4. **Repeat:**
   - Repeat steps 2 and 3 until you reach the end of the array.

5. **One Pass Complete:**
   - After the first pass, the largest element is guaranteed to be at the end of the array.

6. **Repeat for Remaining Elements:**
   - Repeat steps 1-5 for the remaining elements (excluding the last sorted element).

7. **Continue Until Sorted:**
   - Continue the process until the entire array is sorted.

Example:
Consider the array `{5, 2, 8, 1, 6, 3, 7, 4}`.

- Pass 1:
  - Compare and swap (if needed) adjacent elements: `{2, 5, 1, 6, 3, 7, 4, 8}`.
  - Continue until the largest element (8) is at the end.

- Pass 2:
  - Compare and swap adjacent elements: `{2, 1, 5, 3, 6, 4, 7, 8}`.
  - Continue until the second-largest element is in its place.

- Pass 3:
  - Continue this process until the entire array is sorted: `{1, 2, 3, 4, 5, 6, 7, 8}`.

Time Complexity:
The time complexity of the Bubble Sort algorithm is O(n^2), where n is the number of elements in the array. This is because, in the worst case, for each element in the array, you may need to make comparisons and swaps for all other elements.

Space Complexity:
The space complexity of Bubble Sort is O(1) because it only requires a constant amount of additional space for the swapping of elements.

OUTPUT:

```
Original Array:
5 2 8 1 6 3 7 4
Sorted Array (Ascending Order):
1 2 3 4 5 6 7 8
```

==========

20.a)20. (a) Write a Java program to demonstrate different layouts(Flow Layout, Card Layout and Grid Layout)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MovieTicketBookingLayoutDemo {
    private JFrame frame;
    private JPanel mainPanel, flowLayoutPanel, cardLayoutPanel, gridLayoutPanel;
    private CardLayout cardLayout;

    public MovieTicketBookingLayoutDemo() {
        frame = new JFrame("Movie Ticket Booking");
        frame.setSize(600, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        mainPanel = new JPanel(new BorderLayout());
```

```
        // Flow Layout Panel
        flowLayoutPanel = new JPanel(new FlowLayout());
        JComboBox<String> movieComboBox = new JComboBox<>(new String[]{"Avengers:
Endgame", "Spider-Man: No Way Home", "The Matrix Resurrections"});
        flowLayoutPanel.add(new JLabel("Select Movie: "));
        flowLayoutPanel.add(movieComboBox);
        JButton nextButton = new JButton("Next");
        flowLayoutPanel.add(nextButton);

        // Card Layout Panel
        cardLayoutPanel = new JPanel();
        cardLayout = new CardLayout();
        cardLayoutPanel.setLayout(cardLayout);

        JPanel card1 = new JPanel();
        JComboBox<String> showtimeComboBox = new JComboBox<>(new String[]{"10:00
AM", "2:00 PM", "7:00 PM"});
        card1.add(new JLabel("Select Showtime: "));
        card1.add(showtimeComboBox);
        JButton nextButtonCard1 = new JButton("Next");
        card1.add(nextButtonCard1);

        JPanel card2 = new JPanel();
        card2.add(new JLabel("Select Seats: "));
        card2.add(new JCheckBox("Seat 1"));
        card2.add(new JCheckBox("Seat 2"));
        card2.add(new JCheckBox("Seat 3"));
        JButton bookTicketButton = new JButton("Book Ticket");
        card2.add(bookTicketButton);

        cardLayoutPanel.add(card1, "Card1");
        cardLayoutPanel.add(card2, "Card2");

        // Grid Layout Panel
        gridLayoutPanel = new JPanel(new GridLayout(4, 3));
        for (int i = 1; i <= 12; i++) {
            gridLayoutPanel.add(new JButton("Seat " + i));
        }

        // Add components to the main panel
        mainPanel.add(flowLayoutPanel, BorderLayout.NORTH);
        mainPanel.add(cardLayoutPanel, BorderLayout.CENTER);
        mainPanel.add(gridLayoutPanel, BorderLayout.SOUTH);

        // Set up ActionListener for the "Next" button in FlowLayout
        nextButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                cardLayout.next(cardLayoutPanel);
            }
```

```
        });

        // Set up ActionListener for the "Next" button in CardLayout (Card1)
        nextButtonCard1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                cardLayout.next(cardLayoutPanel);
            }
        });

        // Set up ActionListener for the "Book Ticket" button in CardLayout (Card2)
        bookTicketButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                // Handle ticket booking logic here (for demonstration, printing to
console)
                String selectedMovie = (String) movieComboBox.getSelectedItem();
                String selectedShowtime = (String)
showtimeComboBox.getSelectedItem();
                System.out.println("Ticket booked for Movie: " + selectedMovie + ",
Showtime: " + selectedShowtime);
            }
        });

        frame.add(mainPanel);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new MovieTicketBookingLayoutDemo());
    }
}
```

Algorithm:

Certainly! Let's break down the main algorithmic steps of the provided Java program. The program has different components such as JComboBox, JButton, and different layouts like FlowLayout, CardLayout, and GridLayout. The algorithm mainly involves handling user interactions and transitions between different steps of movie ticket booking.

Algorithm:

1. **Create GUI Components:**
   - Initialize JFrame, JPanel, JComboBox, JLabel, JButton, and other necessary components.
   - Set up the initial layout managers for the main panel and sub-panels.

2. **Create Card Layout and Cards:**
   - Initialize a CardLayout for the cardLayoutPanel.
   - Create multiple JPanel instances to represent different "cards" (steps) in the cardLayoutPanel.

   - Add components (JLabel, JComboBox, JButton) to each card as needed.

3. **Add Components to Main Panel:**
   - Add the components to the mainPanel using BorderLayout.
   - FlowLayout is used for the top panel, CardLayout for the center panel, and GridLayout for the bottom panel.

4. **Set Up ActionListeners:**
   - Set up ActionListeners for buttons to handle user interactions.
   - The "Next" button in the FlowLayout and CardLayout switches to the next card using cardLayout.next(cardLayoutPanel).
   - The "Book Ticket" button in CardLayout performs a booking logic (in this case, printing to the console).

5. **Handle Movie Ticket Booking Logic:**
   - When the "Book Ticket" button is clicked, retrieve selected movie and showtime from JComboBoxes.
   - Perform ticket booking logic (for demonstration, print to the console).

6. **Display GUI:**
   - Set the JFrame to be visible.

Example Execution Flow:
   - User selects a movie in the FlowLayout.
   - Clicks "Next" to move to the CardLayout (Card1) where they select a showtime.
   - Clicks "Next" again to move to the CardLayout (Card2) where they can select seats and book a ticket.
   - Clicking "Book Ticket" prints the booked ticket information to the console.

This algorithm provides a high-level overview of the steps involved in creating and interacting with the movie ticket booking GUI using different layouts in the provided Java program.

OUTPUT:

Ticket booked for Movie: Spider-Man: No Way Home, Showtime: 2:00 PM
==========

20.b)Write a Java program to find sum and average of n numbers

```
import java.util.Scanner;

public class SumAndAverage {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input the number of elements
        System.out.print("Enter the number of elements: ");
        int n = scanner.nextInt();
```

```
        // Input the elements
        System.out.println("Enter the elements:");
        int sum = 0;
        for (int i = 0; i < n; i++) {
            System.out.print("Element " + (i + 1) + ": ");
            int num = scanner.nextInt();
            sum += num;
        }

        // Calculate average
        double average = (double) sum / n;

        // Display the sum and average
        System.out.println("Sum: " + sum);
        System.out.println("Average: " + average);

        scanner.close();
    }
}
```

Algorithm:

Input the Number of Elements:

Use Scanner to take the number of elements (n) as input from the user.
Input the Elements:

Use a loop to input each element one by one and update the sum.
Calculate Average:

Calculate the average by dividing the sum by the number of elements (n).
Display Results:

Display the sum and average to the user.
Close Scanner:

Close the Scanner to free up resources.
Compile and run this program to find the sum and average of n numbers entered by the user.

OUTPUT:

```
Enter the number of elements: 5
Enter the elements:
Element 1: 1
Element 2: 2
Element 3: 3
Element 4: 4
```

```
Element 5: 5
Sum: 15
Average: 3.0
================================================================================
============================
```