**Cinema Management System Description**

The provided Java program is a comprehensive Cinema Management System designed to handle various aspects of managing a cinema's seating and ticketing process. This system provides a user-friendly interface for customers and administrators, offering functionalities such as buying and canceling tickets, displaying available seats, and searching for and sorting tickets. Below is a detailed breakdown of the system's features, techniques used, and an overview of its structure and operations.

**System Structure and Operations**

1. \*\*Initialization\*\*:

- The program begins by initializing a 2D array `seats` to represent the seating arrangement in the cinema. Each element in this array is set to 0, indicating that all seats are initially available.

- An array `prices` is defined to store the ticket prices for each row.

- An array `ticketsSold` is used to store information about the tickets that have been sold. Additionally, `ticketCount` keeps track of the total number of tickets sold.

2. \*\*User Interaction\*\*:

- The `main` method contains a menu-driven interface that allows users to interact with the system. This menu offers options to buy a ticket, cancel a ticket, display available seats, find the first available seat, print ticket information and total sales, search for a specific ticket, and sort tickets by price.

3. \*\*Menu Options\*\*:

- \*\*Buy a Ticket\*\*:

- The `buy\_ticket` method prompts the user to enter the row and seat number they wish to book. It validates the input to ensure that the row and seat numbers are within the valid range and that the selected seat is available.

- Upon successful validation, the user is asked to provide their personal information, which is used to create a `Person` object. This information, along with the seat details and price, is used to create a `Ticket` object.

- The ticket is then added to the `ticketsSold` array, the seat is marked as booked, and the ticket information is displayed to the user.

- \*\*Cancel a Ticket\*\*:

- The `cancel\_ticket` method allows the user to cancel a previously booked seat by entering the row and seat number. It checks if the seat is booked, then removes the corresponding ticket from the `ticketsSold` array and marks the seat as available.

- \*\*Display Available Seats\*\*:

- The `print\_seating\_area` method visually represents the seating arrangement, showing available seats with 'O' and booked seats with 'X'.

- \*\*Find the First Available Seat\*\*:

- The `find\_first\_available` method searches the `seats` array for the first available seat and displays its location.

- \*\*Print Ticket Information and Total Sales\*\*:

- The `print\_tickets\_info` method prints the details of all sold tickets and calculates the total sales amount.

- \*\*Search for a Ticket\*\*:

- The `search\_ticket` method allows users to search for a ticket by entering the row and seat number. If a ticket is found, its details are displayed.

- \*\*Sort Tickets by Price\*\*:

- The `sort\_tickets` method sorts the tickets in ascending order based on their price using a bubble sort algorithm and prints the sorted ticket information.

**Techniques Used**

1. \*\*Data Structures\*\*:

- \*\*Arrays\*\*:

- The program uses arrays to manage seating (`seats`), prices (`prices`), and sold tickets (`ticketsSold`). Arrays provide efficient access and manipulation of elements, which is crucial for managing the seating arrangement and ticketing information.

- \*\*Classes and Objects\*\*:

- The program defines `Person` and `Ticket` classes to encapsulate customer and ticket information, respectively. This object-oriented approach helps in organizing and managing data effectively.

2. \*\*Input Validation\*\*:

- Input validation is performed at various stages to ensure that the user inputs valid row and seat numbers. This prevents errors and ensures the integrity of the data.

3. \*\*Sorting Algorithm\*\*:

- A simple bubble sort algorithm is used in the `sort\_tickets` method to sort the tickets by price. This technique, although not the most efficient for large datasets, is suitable for the relatively small number of tickets handled by the system.

4. \*\*User Interface\*\*:

- The menu-driven interface in the `main` method provides a clear and structured way for users to interact with the system. Each menu option corresponds to a specific functionality, making the system intuitive and easy to use.

5. \*\*Error Handling\*\*:

- The program includes basic error handling to manage invalid inputs. For example, if the user enters a non-integer value when a number is expected, the program prompts the user to re-enter the value.