2024

Programming Report

EXPLAINATION ABOUT CODE

ELIAS NADDER

Contents

The App class

The code provided demonstrates the use of several data structures and algorithms. Here are the basic principles of data structures and algorithms used in the code:

1. Arrays: The code uses arrays to store instances of the `ShowTimes` and `Movie` classes. For example, the `showTimes` array is used to store instances of the `ShowTimes` class, and the `movies` array is used to store instances of the Movie class.
2. Lists: The code uses lists to store collections of objects. For example, the `showTimesList` variable is a list that stores instances of the `ShowTimes` class. The showtimes variable in the Movie class is also a list that stores instances of the `ShowTimes` class.
3. Objects: The code defines several classes (`ShowTimes`, `Movie`, `Cinema`, `MyFrame`) to represent different entities in the application. These classes encapsulate data and behavior related to their respective entities.
4. Sublist: The code uses the subList method to create a sublist of `ShowTimes` objects for each movie. This allows each movie to have a unique set of showtimes.
5. String manipulation: The code uses string manipulation techniques to set the titles and genres of the movies. For example, the `setTitle` method in the Movie class sets the title of a movie based on the provided index.
6. Iteration: The code uses loops to iterate over arrays and lists. For example, the for loop in the main method is used to initialize the `showTimes` array and assign unique showtimes to each movie.
7. Conditional statements: The code uses conditional statements (if-else) to determine the showtime and available seats for each `ShowTimes` object. For example, the `decrementAvailableSeats` method in the `ShowTimes` class checks if the available seats are greater than 0 before decrementing them.
8. Event handling: The code implements the ActionListener interface to handle button click events in the `MyFrame` class. This allows the user to interact with the application by reserving tickets, viewing information, signing in, and managing reservations.

Overall, the code demonstrates the use of various data structures (arrays, lists) and algorithms (string manipulation, iteration, conditional statements) to create a movie ticket reservation system. These principles are fundamental to building efficient and organized software applications.

The `MyFrame` class

The `MyFrame` class in the provided code does not explicitly implement any data structures or algorithms. However, it utilizes several Java data structures and algorithms as part of its implementation. Here are some of the basic principles of data structures and algorithms used in the code:

1. Arrays: The code uses arrays to store instances of `JButton`, `ImageIcon`, `String`, and `Movie` objects. Arrays provide a fixed-size collection of elements that can be accessed using an index.
2. Lists: The code uses `List<Ticket>` to store a collection of Ticket objects. Lists are dynamic data structures that can grow or shrink in size as needed.
3. Objects: The code defines several classes (`User`, `AccountDetails`, `Ticketing`, etc.) to represent real-world entities and their properties. Objects are instances of these classes that encapsulate data and behavior.
4. Inheritance: The `MyFrame` class extends the `JFrame` class, inheriting its properties and methods. Inheritance is a mechanism that allows a class to inherit the characteristics of another class, promoting code reuse and creating a hierarchical relationship between classes.
5. Polymorphism: The `MyFrame` class implements the ActionListener interface and overrides its `actionPerformed` method. Polymorphism allows objects of different classes to be treated as

objects of a common superclass or interface, enabling method overriding and dynamic method dispatch.
6. Control Structures: The code uses control structures such as if statements and for loops to control the flow of execution based on certain conditions. Control structures are used to make decisions and repeat actions based on specific criteria.
7. Event Handling: The `MyFrame` class implements the ActionListener interface to handle user actions such as button clicks. Event handling is a mechanism that allows the program to respond to user interactions or system events.
8. Layout Managers: The code uses layout managers (`BorderLayout`, `GridLayout`, `FlowLayout`) to arrange and position GUI components within containers. Layout managers provide a way to control the size and position of components in a flexible and platform-independent manner.

These are some of the basic principles of data structures and algorithms used in the provided code. It's important to note that the code does not explicitly implement complex algorithms or data structures, but rather utilizes built-in Java data structures and algorithms to achieve its functionality.

The reserveWindow class

The code snippet provided does not explicitly show the use of any data structures or algorithms. However, we can infer the following principles of data structures and algorithms being used:

1. Object-Oriented Programming: The code follows the principles of object-oriented programming by defining classes (`ReserveWindow`, `Movie`, `User`, `Ticket`, `Ticketing`) and creating objects of these classes to represent real-world entities and their interactions.
2. Arrays: The code uses arrays to store and manipulate multiple instances of `JRadioButton` objects (`showTimeButton`) and Ticket objects (`tickets`).
3. Lists: The code uses the List interface to store and manipulate a collection of Ticket objects (`tickets`).
4. Conditional Statements: The code uses conditional statements (if-else) to make decisions based on certain conditions. For example, it determines the value of `cinemLabel` based on the value of k.
5. Loops: The code uses loops (for) to iterate over arrays and perform actions on each element. For example, it iterates over the `showTimeButton` array to add action listeners to each button.
6. File I/O: The code uses file input/output operations to write ticket information to a file. The `writeToStringFile` method uses a `BufferedWriter` to write the ticket information to a file specified by the filename parameter.

Overall, the code snippet demonstrates the use of basic data structures like arrays and lists, as well as the implementation of conditional statements, loops, and file I/O operations. However, it does not showcase any advanced data structures or algorithms.

The ReservedTicketWindow class

The `ReservedTicketWindow` class does not explicitly implement any data structures or algorithms. However, it does make use of some basic data structures and algorithms indirectly.

1. Arrays: The class uses arrays to store and display the reserved tickets. The `ticketLabel` array is used to store and display the ticket information for each reserved ticket. The size of the array is determined by the number of reserved tickets. The `ticketPanel` array is used to create panels for each reserved ticket.
2. Lists: The class uses lists to retrieve ticket information from the Ticketing object. The `ticketing.getTickets()` method returns a list of reserved tickets, which is used to iterate over and display the ticket information.
3. String manipulation: The class uses string manipulation to display the ticket information. The `toString()` method of the Ticketing object is called to convert the ticket information into a string representation, which is then displayed using the `JLabel` component.

4. Control flow: The class uses control flow statements such as if-else and for loops to handle different scenarios. For example, if there are no reserved tickets, a message is displayed indicating that there are no reserved tickets. If there are reserved tickets, the ticket information is displayed using a loop.

Overall, the `ReservedTicketWindow` class demonstrates the use of basic data structures like arrays and lists, as well as simple algorithms like iteration and string manipulation, to display reserved ticket information in a graphical user interface.

The CancelReservationWindow class

The `CancelReservationWindow` class does not explicitly use any data structures or algorithms. However, it does interact with the Ticketing class, which may use data structures and algorithms internally.

Based on the provided code, it seems that the Ticketing class is responsible for managing reservations and cancellations. It is not clear from the code snippet what specific data structures or algorithms are used in the Ticketing class.

To provide a more detailed report on the data structures and algorithms used, it would be necessary to analyze the implementation of the Ticketing class.

The CreditWindow class

The `CreditWindow` class in the provided Java code does not explicitly use any data structures or algorithms. However, it does make use of some basic principles related to data structures and algorithms.

1. Data Structure:
    - The class `CreditWindow` uses instance variables to store different components of the GUI, such as `JFrame`, `JLabel`, `JTextField`, `JButton`, and `JPanel`. These variables are used to create and manipulate the graphical user interface.
    - The User class, which is passed as a parameter to the `CreditWindow` constructor, is assumed to have its own data structure to store user information, such as `AccountDetails`.
2. Algorithm:
    - The `CreditWindow` class implements the ActionListener interface and overrides the `actionPerformed` method. This method is called when the user interacts with the submit button.
    - Inside the `actionPerformed` method, there is a conditional statement that checks if the submit button was clicked. If it was, it performs the following steps:
        - It checks if the `accountNum` text field is not null.
        - If the `accountNum` is not null, it creates an instance of the AccountDetails class and sets the account number using the value entered in the accountNum text field.
        - It then sets the `AccountDetails` object as a property of the User object.
        - Finally, it disposes of the `CreditWindow` frame.

In summary, the `CreditWindow` class demonstrates the use of basic data structures (instance variables) and implements a simple algorithm (event handling) to capture user input and update the user's account details.

The infoNewWindow class

The `infoNewWindow` class does not explicitly implement any data structures or algorithms. However, it utilizes several Java Swing components and follows some basic principles of object-oriented programming.

1. Data Structures:
    - `JFrame`: Represents the main window of the application.
    - `JButton`: Represents a button component.
    - `JLabel`: Represents a label component.
    - `List<Ticket>`: Represents a list of tickets.
    - `Movie`: Represents a movie object.
    - `User`: Represents a user object.
    - `Ticketing`: Represents a ticketing object.
2. Algorithms:
    - `actionPerformed(ActionEvent e)`: This method is an event handler that is triggered when the reserve button is clicked. It checks if the user is signed in and creates a new `ReserveWindow` object if the user is signed in. Otherwise, it displays an error message.
3. Object-Oriented Principles:
    - Encapsulation: The class encapsulates the `movie`, `path`, `user`, `k`, `tickets`, and `ticketing` objects as private instance variables.
    - Inheritance: The class extends the Thread class and implements the ActionListener interface.
    - Polymorphism: The `actionPerformed` method is an example of polymorphism, as it can handle different types of events.
    - Composition: The class composes multiple Swing components to create the user interface.
    - Abstraction: The class abstracts the details of creating and managing the second window by encapsulating the logic in a separate class.

Overall, the `infoNewWindow` class demonstrates the use of data structures and algorithms indirectly through the utilization of Java Swing components and basic object-oriented programming principles.

The Ticketing class

The `Ticketing` class in the provided code uses several data structures and algorithms to manage ticket reservations. Here are the basic principles of data structures and algorithms used in the class:

1. List: The Ticketing class uses the `List<Ticket>` data structure to store a collection of Ticket objects. This allows for easy manipulation and retrieval of tickets.
2. Iterator: The class uses an Iterator to safely remove items from the tickets list. The iterator allows for efficient traversal and removal of elements from the list.
3. Conditional Statements: The class uses conditional statements (if and else) to perform checks and make decisions based on certain conditions. For example, it checks if a payment method is selected before processing a ticket reservation.
4. Looping: The class uses loops (for and while) to iterate over elements in the tickets list and perform operations on them. For example, it iterates over the list to find and remove a ticket with a specific ID.
5. Object Composition: The class uses composition to create instances of other classes, such as `User`, `Movie`, and `ShowTimes`. This allows for the organization and management of related data.

6. Encapsulation: The class encapsulates data by providing getter and setter methods for accessing and modifying the attributes of the Ticketing object. This promotes data integrity and encapsulation.
7. Error Handling: The class uses `JOptionPane` to display error messages and information to the user in case of invalid inputs or errors during ticket reservation or cancellation.

Overall, the Ticketing class demonstrates the use of various data structures and algorithms to manage ticket reservations efficiently. It utilizes lists, iterators, conditional statements, loops, object composition, encapsulation, and error handling to provide a robust ticketing system.

The Ticket class

The `Ticket` class does not explicitly use any data structures or algorithms. It is a simple class that represents a ticket for a show. It has instance variables to store the seat number, showtime, ticket price, and ID. It also has getter and setter methods for each of these variables.

Data structures and algorithms are typically used to solve more complex problems and perform operations on data. In this case, the Ticket class does not require any complex data structures or algorithms. It mainly focuses on storing and retrieving data related to a ticket.

However, if you were to use this Ticket class in a larger system, you might need to use data structures and algorithms to perform operations such as searching for tickets, sorting tickets based on certain criteria, or managing a collection of tickets efficiently. In such cases, you could use data structures like arrays, lists, or maps, and algorithms like sorting or searching algorithms to achieve the desired functionality.

In summary, while the Ticket class itself does not use any data structures or algorithms, they may be required in a larger system that uses this class to perform more complex operations on tickets.

The Cinema class

The `Cinema` class in the provided Java code does not explicitly demonstrate the use of data structures and algorithms. However, it does utilize a data structure, specifically the List interface from the `java.util package`, to store a collection of `Movie` objects.

Data structures are essential for organizing and managing data efficiently. In this case, the `List<Movie>` movies attribute serves as a data structure to store multiple Movie objects. The List interface provides methods to add, remove, and access elements in the collection.

Algorithms, on the other hand, are not explicitly demonstrated in the given code snippet. Algorithms are step-by-step procedures used to solve problems or perform specific tasks. While the code does not contain any explicit algorithms, it is important to note that algorithms can be implemented in methods or operations performed on the data structure.

For example, if you were to add a method to search for a specific movie in the movies list, you could implement an algorithm such as linear search or binary search to efficiently locate the desired movie.

In summary, the Cinema class demonstrates the use of a data structure (List) to store a collection of Movie objects. However, it does not explicitly showcase any algorithms. Algorithms can be implemented in methods or operations performed on the data structure to solve specific problems or perform tasks efficiently.

The Movie class

The `Movie` class in the provided code uses a few basic principles of data structures and algorithms. Here are the key points:

1. Data Structure: The class uses a `List<ShowTimes>` to store the showtimes of the movie. This allows for multiple showtimes to be associated with a single movie. The List data structure provides dynamic resizing and efficient access to elements.

2. Algorithm: The `printShowTime` method uses an algorithm to retrieve the showtime of a movie at a specific index. It accesses the showtimes list and retrieves the showtime using the `getShowTime` method of the `ShowTimes` class.
3. Algorithm: The `printSeats` method uses an algorithm to retrieve the available seats for a movie at a specific index. It accesses the showtimes list and retrieves the available seats using the `getAvailableSeats` method of the `ShowTimes` class.

Overall, the code demonstrates the use of a data structure (List) to store and retrieve showtimes, and algorithms to access specific showtimes and available seats. These principles are fundamental to working with data structures and algorithms in Java.

The User class

The given `User` class does not explicitly use any data structures or algorithms. However, it does have a composition relationship with the `AccountDetails` class, which may internally use data structures and algorithms.

Data structures are used to organize and store data efficiently, while algorithms are used to perform operations on that data. Some common data structures include arrays, linked lists, stacks, queues, and trees. Algorithms can be used to search, sort, insert, delete, or manipulate data within these data structures.

In the User class, the `username` variable is a simple string data type, which does not require any specific data structure. It is used to store the username of the user.

The `accountDetails` variable is of type `AccountDetails`, which is not provided in the given code. Without knowing the implementation details of the `AccountDetails` class, it is difficult to determine the specific data structures and algorithms used within it.

To provide a more detailed report on the data structures and algorithms used, it would be necessary to analyze the implementation of the `AccountDetails` class.

The ShowTimes class

The `ShowTimes` class does not explicitly use any data structures or algorithms. However, it does provide a basic example of encapsulation and object-oriented programming principles.

Encapsulation is demonstrated through the use of private instance variables (`showTime` and `availableSeats`) and public getter and setter methods. This allows for controlled access to the class's internal state, promoting data integrity and encapsulation.

Object-oriented programming principles are also evident in the class's constructor and methods. The constructor allows for the creation of `ShowTimes` objects with initial values for `showTime` and `availableSeats`. The getter and setter methods provide access to these values, allowing for manipulation and retrieval of the object's state.

While the class itself does not directly implement any complex data structures or algorithms, it serves as a foundation for potential future enhancements. For example, additional methods could be added to implement more advanced algorithms, such as searching for specific show times or optimizing seat allocation.

In summary, the `ShowTimes` class demonstrates the principles of encapsulation and object-oriented programming. While it does not currently utilize any advanced data structures or algorithms, it provides a solid foundation for potential future enhancements.

The class AccountDetails

The class `AccountDetails` does not explicitly use any data structures or algorithms. It is a simple Java class that represents an account with three attributes: `name`, `email`, and `accountNum`.
The class provides basic getter and setter methods for each attribute, allowing access and modification of the account details. These methods follow the principle of encapsulation, which is a fundamental concept in object-oriented programming.
In terms of data structures, the class uses three private instance variables of type String to store the account details. These variables are used to hold the name, email, and account number of an account. However, the class does not utilize any complex data structures such as arrays, lists, or maps.
As for algorithms, the class does not implement any specific algorithms. It mainly focuses on providing methods to get and set the account details. These methods perform simple operations such as returning the value of an attribute or updating its value.
In summary, the `AccountDetails` class does not involve any advanced data structures or algorithms. It serves as a basic representation of an account and provides methods to access and modify its attributes.