1. Admin

A number of essential Object-Oriented Programming (OOP) ideas are used in the code sample. The OOP ideas used in the code are as follows:

Class: The code designates the Admin class, which acts as a template for producing admin objects. It combines related data (instance variables) and action (methods) into a single thing.

Encapsulation: The instance variables of the Admin class (name, username, email, phone, password) are defined as private, hiding any implementation-specific information. Getter methods (getName(), getUsername(), getEmail(), getPhone(), and getPassword()) are used to gain access to these variables, encouraging encapsulation and data abstraction.

Constructor: An initialization of an Admin object with particular settings is possible using the class's constructor Admin(). To create and initialise objects, constructors are needed.

Information Hiding: The instance variables (name, username, email, phone, password) have private access modifiers set to them so that outside of the class, they cannot be directly accessed. Through getter methods, this ensures data encapsulation and information masking while enabling restricted access to the data.

The class has a method called isValidLogin(String inputUsername, String inputPassword) that verifies the accuracy of login information. The actions and behaviours ascribed to objects are specified by methods.

Object: Admin class instances represent distinct admin entities, each with an own collection of data. Classes are used to build objects, and objects communicate with one another by calling methods and accessing properties.

These OOP ideas support abstraction, encapsulation, reuse, and code organisation. The code accomplishes modularity through the use of classes, objects, and encapsulation, enabling concern separation and simpler maintenance. By modelling entities as objects with their own characteristics and behaviours, the code exemplifies the fundamental ideas behind OOP.

name: Displays the admin's name.

username: Displays the administrator's username.

email: Displays the administrator's email address.

phone: Displays the administrator's phone number.

password: Displays the administrator's password.

These variables are sent as arguments to the class's constructor, which initialises the associated instance variables.

In order to access the values of the instance variables, the class additionally includes getter methods (getName(), getUsername(), getEmail(), getPhone(), and getPassword()).

The class also has a method called isValidLogin(String inputUsername, String inputPassword) that verifies the login credentials supplied against the admin's username and password. If the values match, indicating a legitimate login, it returns true; if not, it returns false.

Overall, this class provides methods to access the admin's information and verify login credentials while representing an admin with basic information.

1. Booking

The BookingRoom class serves to represent a reservation for a room at a hotel or other type of housing. It saves details about the reservation, including the reservation ID, the booked start and end dates, the room ID, the customer ID, the capacity, the status, and the price.

OOP Concepts: a. Encapsulation: By offering private fields and public getter and setter methods, the class encapsulates the data associated with a reservation. By doing this, data integrity is ensured, and access to and modification of the booking information are controlled.

b. Constructor: The class has a constructor that enables initialization of all booking information at the moment an object is created. This makes it easier to create a booking instance with the necessary data.

c. Getters and Setters: By providing getter and setter methods for each field, the class enables other classes to access and alter the booking information in a controlled way. This encourages data abstraction and encapsulation.

Expected Output: The BookingRoom class doesn't provide any output on its own. It serves as a guide for making items that stand in for certain reservations. This class's output or application will rely on how it is applied inside a bigger programme. For instance, other classes might utilise this class to create and manage reservations, obtain reservations data, or do computations based on the reservations information.

1. Login

The Homepage class serves as the application's core framework and provides the UI elements for the login screen. Users are able to do things depending on the login state by entering their username and password, clicking the login button, and so on.

Concepts from OOP: 1. Inheritance The Javax.swing is extended by the Homepage class.JFrame class, a Swing component, is used to build the application's window frame. It takes on the characteristics and actions of the JFrame class.

b. Construction: Labels, text fields, buttons, and panels are just a few of the Swing elements included in the Homepage class. The graphical user interface is made up of these parts together.

c. Encapsulation: By designating the UI elements as private fields, the class encapsulates them. Then, using the proper methods like readAdminsFromFile(), isValidLogin(), and event handlers like jButton2ActionPerformed() and jButton1ActionPerformed(), these fields are accessed and modified.

d. Event-driven programming: To react to user activities, the code makes use of event listeners and handlers. For instance, when the "Register now" button is pressed, the jButton2ActionPerformed() function is called, and when the "Login" button is hit, the jButton1ActionPerformed() method is performed.

Expected Output: Because the code simulates the GUI's structure and behaviour, it is unable to provide any specific output on its own. The exact result would depend on how the user utilised and interacted with the programme. The "Register now" button activates a new registration window (Register class) when it is clicked. When the "Login" button is hit, it checks the admin information from a file, validates the provided username and password, and displays a success or error message as necessary.

Overall, the Homepage class shows how to implement a login screen in a Java Swing application using OOP principles like inheritance, composition, encapsulation, and event-driven programming.

1. Regitser

The Register class serves as a representation of the registration screen's frame. Users can input personal data including their entire name, username, email, phone number, password, and confirmation password. The class also has methods for validating each field, as well as one for writing the registration data to a file.

OOP Concepts: a. Encapsulation: By designating the UI components as private fields, the class encapsulates them. Additionally, it offers private methods for database operations and field validation.

b. Inheritance: The Register class inherits the JFrame class's features and functionality by extending the javax.swing.JFrame class.

c. Makeup: The class includes a variety of Swing elements, including labels, text fields, and password fields. The graphical user interface is made up of these parts together.

Field Validation: The class has a number of private methods for verifying various fields, including:

The complete name is verified using validateFullName().

The validateUsername() function verifies the username and looks for duplicates.

The email address is verified by validateEmail().

The phone number is verified by validatePhoneNumber().

The password is verified by validatePassword().

By contrasting it with the supplied password, the validateConfirmationPassword() function verifies the confirmation password.

Operation of databases:

The registration information is written to a file called "Admin.txt" using the putAdmintoDatabase() function. Using the Generator, it pulls a value from the file.If the file doesn't already exist, the readFirstValue() function creates it and appends the registration information to it.

Expected Output: Because the code simulates the GUI's structure and behaviour, it is unable to provide any specific output on its own. The exact result would depend on how the user utilised and interacted with the programme. The registration information will be written to the "Admin.txt" file after providing the necessary information and selecting the relevant button; a success or error message will then be shown.

Overall, the Java Swing application's registration page is created using OOP principles like encapsulation, inheritance, and composition thanks to the Register class. To manage user input and save the registration data, it contains field validation techniques and a database operation method.

Event handlers for a number of the elements on the registration screen are included in the additional code you offered. Let's go through each event handler and describe what it does:

jTextField5ActionPerformed:

When an action (such as hitting Enter) takes place in the jTextField5 component, which stands in for the phone number field, this event handler is called.

To verify the user-entertainment of a phone number, it invokes the validatePhoneNumber() function.

jTextField4ActionPerformed:

When something happens in the jTextField4 component, which stands in for the username field, this event handler is called.

To verify the username that the user submitted, it invokes the validateUsername() function.

jTextField3ActionPerformed:

When something happens in the jTextField3 component, which stands in for the email field, this event handler is invoked.

To verify the email address, it uses the validateEmail() function.

jPasswordField3ActionPerformed:

When something happens in the jPasswordField3 component, which stands in for the password field, this event handler is invoked.

To verify the user-provided password, it invokes the validatePassword() function.

jButton5ActionPerformed:

When a user hits the button-like jButton5 component, this event handler is activated.

A new instance of the Homepage class is created, its visibility is set, the existing registration frame is disposed of, and the frame is closed.

jTextField2ActionPerformed:

When something happens in the jTextField2 component, which stands in for the complete name field, this event handler is invoked.

To verify the user's complete name entered, it invokes the validateFullName() function.

1. Generator

The use of object-oriented programming (OOP) principles is demonstrated in the given code. Let's talk about the OOP ideas utilised in the code and how they affect how it functions.

Classes: To encapsulate the ID generating logic, the code constructs a class called Generator. The class has methods that may be used to carry out particular activities.

There are two methods in the class: readFirstValue and generateNextID. These methods can be separately invoked to carry out the functions that they each encapsulate.

Encapsulation: The class encapsulates the relevant functionality of ID development, hide the implementation specifics from external programmes. The methods offer a means of gaining access to and modifying the class's internal state and behaviour.

ReadFirstValue and generateNextID are both specified as static methods. This indicates that they may be reached and used without first constructing a Generator class object. Static methods are part of the class itself, not a class instance.

Exception Handling: The try-catch construct is used in the code to show exception handling. Any potential IOException that could arise when reading a file is caught, and an error message is shown.

File Handling: Using the BufferedReader class, the code uses file handling methods to read a file's contents. It examines the file line by line after opening it before extracting the required information.

String manipulation is used by the code to extract data from each line and create the subsequent ID dependig on the preceding ID.

Return Values: The procedures generateNextID and readFirstValue provide string values. The created IDs are represented by these return values.

The methods generateNextID and readFirstValue both take input arguments (filename, value, and latestID). These arguments enable the methods to get the data they need to function.

The code that calls these methods receives the created ID back as a string value, which may be utilised or displayed. Depending on its particular needs, the calling code might decide how to handle or display the produced ID.

In conclusion, the example code illustrates OOP concepts by using static methods, managing exceptions, encapsulating functionality into classes, and giving methods with input arguments and return values. It makes it possible to create new IDs based on older IDs that have been read from a file, giving you flexibility and reuseability when creating new IDs.

1. Customer

A Customer class is represented by the code given. Let's examine the code to determine its goal, the OOP principles used, and the methods it offers.

Customer is a class that the code defines that contains the characteristics and behaviours connected to a customer.

Instance Variables: The class defines a number of private instance variables, including hotelcount, customerID, name, email, and phone numbers. The specific information of a customer object is stored in these variables.

Customer has a constructor that accepts the following parameters: customerID, name, IC, passport, email, phone, and hotelcount. The initialization of the instance variables with the supplied values is the responsibility of the constructor.

Encapsulation: Because the instance variables are designated as private, only members of the Customer class may access or change them. As a result, other code cannot directly access these variables, ensuring data integrity and encapsulation.

Getter Methods: For each instance variable, the class has a getter method. The values of the instance variables can be retrieved by external code using these methods, which include getCount(), getPassport(), getId(), getName(), getEmail(), getPhone(), and getIC(). By making these methods available, encapsulation is encouraged while controlling access to the instance variables.

Currently, the class only has getter methods, which provide users access to the contents of instance variables. You may add more instance methods to the client class to carry out tasks like updating client information, computing statistics, etc

The code focuses on creating a Customer object that contains getter methods for accessing the customer data. This encapsulation supports the preservation of data integrity and permits managed access to consumer information. Depending on the demands of the application, more methods and behaviours may be added to the class to improve its usefulness.

Note: The sample code does not have any primary methods or code that would show how to use the Customer class. The class itself serves as a model or template for building specific customer objects, and it may be used to a bigger programme that needs to manage customer data