

|  |  |
| --- | --- |
| import javax.swing.\*;  class User {  String userId;  String password;  1. public User(String userId, String password) {  this.userId = userId;  this.password = password;  }  }  2. class BSTNode {  User user;  BSTNode left, right;  public BSTNode(User user) {  this.user = user;  left = right = null;  }  }  3. public class LoginValidationSystem {  private BSTNode root;  public LoginValidationSystem() {  root = null;  }  4. public void insert(User user) {  root = insertRec(root, user);  }  5. private BSTNode insertRec(BSTNode root, User user) {  6. if (root == null) {  root = new BSTNode(user);  return root;  }  if (user.userId.compareTo(root.user.userId) < 0) {  root.left = insertRec(root.left, user);  } else if (user.userId.compareTo(root.user.userId) > 0) {  root.right = insertRec(root.right, user);  }  return root;  }  7. public boolean search(String userId) {  return searchRec(root, userId);  }  8. private boolean searchRec(BSTNode root, String userId) {  9. if (root == null) {  return false;  }  if (userId.compareTo(root.user.userId) == 0) {  return true;  } else if (userId.compareTo(root.user.userId) < 0) {  return searchRec(root.left, userId);  } else {  return searchRec(root.right, userId);  }  }  10. public boolean validatePassword(String userId, String password) {  return validatePasswordRec(root, userId, password);  }  11. private boolean validatePasswordRec(BSTNode root, String userId, String password) {  12. if (root == null) {  return false;  }  if (userId.compareTo(root.user.userId) == 0) {  return password.equals(root.user.password);  } else if (userId.compareTo(root.user.userId) < 0) {  return validatePasswordRec(root.left, userId, password);  } else {  return validatePasswordRec(root.right, userId, password);  }  }  13. public static void main(String[] args) {  LoginValidationSystem loginSystem = new LoginValidationSystem();  14. while (true) {  // Create a panel for user selection  15. JPanel selectionPanel = new JPanel();  JRadioButton createAccountButton = new JRadioButton("Create Account"); JRadioButton loginButton = new JRadioButton("Login");  ButtonGroup group = new ButtonGroup();  group.add(createAccountButton);  group.add(loginButton);  selectionPanel.add(createAccountButton);  selectionPanel.add(loginButton);  16. int selectionResult = JOptionPane.*showConfirmDialog*(null, selectionPanel, "Choose an Option",  JOptionPane.*OK\_CANCEL\_OPTION*, JOptionPane.*PLAIN\_MESSAGE*);  17 if (selectionResult == JOptionPane.*OK\_OPTION*) {  if (createAccountButton.isSelected()) {  // Create Account  JTextField userIdField = new JTextField(20);  JPasswordField passwordField = new JPasswordField(20);  JPanel accountCreationPanel = new JPanel();  accountCreationPanel.add(new JLabel("User ID:"));  accountCreationPanel.add(userIdField);  19.accountCreationPanel.add(new JLabel("Password:"));  accountCreationPanel.add(passwordField);  int accountCreationResult = JOptionPane.*showConfirmDialog*(null, accountCreationPanel, "Create Account",  JOptionPane.*OK\_CANCEL\_OPTION*, JOptionPane.*PLAIN\_MESSAGE*);  if (accountCreationResult == JOptionPane.*OK\_OPTION*) {  String userId = userIdField.getText();  String password = new String(passwordField.getPassword());  // Insert the user into the system  loginSystem.insert(new User(userId, password));  21.JOptionPane.*showMessageDialog*(null, "Account created successfully. Please login.");  }  22. else {  JOptionPane.*showMessageDialog*(null, "Account creation canceled");  }  }  23. else if (loginButton.isSelected()) {  // Login  JTextField userIdField = new JTextField(20);  JPasswordField passwordField = new JPasswordField(20);  24. JPanel loginPanel = new JPanel();  loginPanel.add(new JLabel("User ID:"));  25.loginPanel.add(userIdField);  loginPanel.add(new JLabel("Password:"));  loginPanel.add(passwordField);  int loginResult = JOptionPane.*showConfirmDialog*(null, loginPanel, "Login",  JOptionPane.*OK\_CANCEL\_OPTION*, JOptionPane.*PLAIN\_MESSAGE*);  if(loginResult ==JOptionPane.*OK\_OPTION*) {  String userId = userIdField.getText();  String password = new String(passwordField.getPassword());  26. // Check if the user exists  if (loginSystem.search(userId)) {  27. // Validate password  if (loginSystem.validatePassword(userId, password)) {  28.JOptionPane.*showMessageDialog*(null, "Login successful. Exiting program.");  return; // Terminate the program upon successful login  } else {  30.JOptionPane.*showMessageDialog*(null, "Invalid password");  }  } else {  31.JOptionPane.*showMessageDialog*(null, "User not found. Create an account first.");  }  } else {  29.JOptionPane.*showMessageDialog*(null, "Login canceled");  }  }  } else {  32.JOptionPane.*showMessageDialog*(null, "Operation canceled. Exiting program.");  return; // Terminate the program if the user cancels the operation  }  }  }  } | 1. The `User` class represents a user with `userId` and `password` attributes.  2. The `BSTNode` class represents a node in a binary search tree (BST) with a `User` object and references to left and right child nodes.  3. The `LoginValidationSystem` class is the main class that contains the logic for inserting users into the BST.  4. In the `insert` method, a new user is inserted into the BST using the `insertRec` method.  5. The `insertRec` method recursively inserts a new user into the BST based on the comparison of user IDs. If the user ID is less than the current node's user ID, it is inserted into the left subtree; if it is greater, it is inserted into the right subtree.  6. The `root` of the BST is updated after each insertion.  7. The `search` method is used to search for a specific user in the BST based on the user ID.  8. The `searchRec` method is a recursive helper function that performs the actual search operation in the BST.  9. In the `searchRec` method:  - If the current node is null (reached a leaf node), it means the user with the given user ID does not exist in the BST, so it returns false.  - If the user ID matches the user ID of the current node, it returns true, indicating that the user with the given user ID is found.  - If the user ID is less than the user ID of the current node, the search continues in the left subtree.  - If the user ID is greater than the user ID of the current node, the search continues in the right subtree.  10. The `validatePassword` method is used to validate the password for a specific user in the BST based on the user ID and password provided.  11. The `validatePasswordRec` method is a recursive helper function that performs the password validation operation in the BST.  12. In the `validatePasswordRec` method:  - If the current node is null, it means the user with the given user ID does not exist in the BST, so it returns false.  - If the user ID matches the user ID of the current node, it checks if the provided password matches the stored password for that user. It returns true if the passwords match, indicating successful validation.  - If the user ID is less than the user ID of the current node, the validation continues in the left subtree.  - If the user ID is greater than the user ID of the current node, the validation continues in the right subtree.    13. The `main` method initializes the `LoginValidationSystem` for user account management.  14. It enters an infinite loop using `while (true)` to continuously prompt the user for actions.  15. Within the loop, it creates a panel for user selection, which includes options to "Create Account" or "Login" using radio buttons (`JRadioButton`) added to a `ButtonGroup`.  16. It then displays a dialog box using `JOptionPane.showConfirmDialog` to prompt the user to choose an option.  17. If the user selects "Create Account," it prompts the user to enter a user ID and password using text fields (`JTextField`) and a password field (`JPasswordField`) within another panel.  18. It then displays another dialog box to confirm the account creation.  19. If the user selects "Create Account" and confirms the account creation, the code retrieves the entered user ID and password from the text fields and password field.  20. It then calls the `insert` method of the `loginSystem` to insert a new `User` object with the provided user ID and password into the system.  21. After successful insertion, it displays a message using `JOptionPane.showMessageDialog` to inform the user that the account has been created successfully.  22. If the user cancels the account creation, it displays a message indicating that the account creation has been canceled.  23. If the user selects "Login," it prompts the user to enter their user ID and password using text fields and a password field within another panel.  24. It then displays a dialog box to confirm the login.  25. If the user confirms the login, the code retrieves the entered user ID and password.  26. It checks if the user exists in the system by calling the `search` method of the `loginSystem`.  27. If the user exists, it validates the password by calling the `validatePassword` method of the `loginSystem`.  28. If the password is valid, it displays a message indicating that the login was successful using `JOptionPane.showMessageDialog`, and then terminates the program by returning from the `main` method.  29. If the user cancels the login, it displays a message using `JOptionPane.showMessageDialog` indicating that the login has been canceled.  30. If the user exists in the system but the entered password is invalid, it displays a message using `JOptionPane.showMessageDialog` indicating that the password is invalid.  31. If the user does not exist in the system, it displays a message using `JOptionPane.showMessageDialog` indicating that the user was not found and prompts the user to create an account first.  32. If the user cancels the operation (selects "Cancel" in the initial selection panel), it displays a message using `JOptionPane.showMessageDialog` indicating that the operation has been canceled and then terminates the program by returning from the `main` method. |

1. **BSTNode Class:**

- The `BSTNode` class represents a node in the Binary Search Tree (BST).

- It has three attributes: `userId` (String) to store the user ID, `password` (String) to store the password, and references `left` and `right` to the left and right child nodes, respectively.

- The constructor initializes the node with the given `userId` and `password`, setting the child nodes to null initially.

2. **LoginValidationSystem Class:**

- This class manages the Binary Search Tree (BST) for user validation.

- It has a private attribute `root` of type `BSTNode` representing the root of the BST.

3. **insertUser Method:**

- `insertUser(String userId, String password)` method inserts a new user into the BST.

- It calls the private helper method `insert` to recursively insert the user into the correct position in the BST.

**4. insert Method:**

- `insert(BSTNode root, String userId, String password)` is a private helper method for inserting a user into the BST.

- It recursively traverses the BST to find the correct position based on the `userId` comparison and inserts the new user node accordingly.

- If the `root` is null, it creates a new node for the user.

**5. searchUser Method:**

- `searchUser(String userId, String password)` method checks if a user with the given `userId` and `password` exists in the BST.

- It calls the private helper method `search` to perform a recursive search in the BST.

**6. search Method:**

- `search(BSTNode root, String userId, String password)` is a private helper method for searching a user in the BST.

- It recursively traverses the BST based on `userId` comparison and checks if the user node with matching `userId` and `password` exists.

**7. Main Method:**

- The `main` method demonstrates the usage of the `LoginValidationSystem` class.

- It creates an instance of `LoginValidationSystem` and inserts several users into the BST using the `insertUser` method.

- Then, it validates login credentials for different users using the `searchUser` method and prints the results (true/false) based on the existence of the user with matching credentials.