Algorithm

**Soil\_Enthusiast\_Home.java:**

* + Creates the main GUI window with image labels and buttons for website demos, videos, and login.

**SoilEnthusiast.java:**

* + Executes commands related to database setup and user registration.

**DatabaseSetup.java:**

* + Establishes a connection to a MySQL database and creates required databases and tables.

**TableSetup.java:**

* + Establishes a connection to a MySQL database and creates necessary tables.

**UserRegistrationForm.java:**

* + Creates a user registration form with input fields and validation.

**LoginFrame.java:**

* + Creates a login form for user authentication.

**UserDashboard.java:**

* + Implements the user dashboard with product categories, cart functionality, checkout, and compost calculator.

**CompostCalculatorFrame.java:**

* + Creates a frame for compost calculation based on user inputs.

**BillingDashboard.java:**

* + Creates a billing dashboard for cart summary and payment options.

**PaymentOptionsFrame.java:**

* + Creates a frame for payment options with validation and order details processing.

File Name: Soil\_Enthusiast\_Home.java

1. Start the Soil\_Enthusiast\_Home class which extends JFrame.
2. Set the title of the JFrame to "Soil Enthusiast".
3. Set the default close operation to EXIT\_ON\_CLOSE.
4. Set the layout of the JFrame to a 4x1 grid layout.
5. Create an ImageIcon by resizing the "welcome\_image.jpg" to 65% of its original size and create a JLabel with this ImageIcon. Add the JLabel to the JFrame.
6. Create buttonPanel1 with a 1x2 grid layout and a horizontal and vertical gap of 20 pixels.
7. Resize "website\_image.png" to 70% of its original size and "website\_video.png" to 80% of its original size to create ImageIcons.
8. Create websiteButton with text "Website Demo" and the resized websiteImageIcon. Add an ActionListener to open the webpage "<https://soil-enthusiast-2024.netlify.app/>" when clicked.
9. Create websiteVideoButton with text "Website Video" and the resized websiteVideoIcon. Add an ActionListener to open the webpage "<https://youtu.be/vrqANy-PgS0>" when clicked.
10. Create buttonPanel2 with a 1x2 grid layout and a horizontal and vertical gap of 20 pixels.
11. Resize "app\_image.jpg" and "app\_video.jpg" to 70% of their original size to create ImageIcons.
12. Create appButton with text "App Demo" and the resized appImageIcon. Add an ActionListener to open the webpage "<https://youtu.be/j24O1qllG2o>" when clicked.
13. Create appVideoButton with text "App Video" and the resized appVideoIcon. Add an ActionListener to open the webpage "<https://youtu.be/Y3IPMGGflmY>" when clicked.
14. Add appVideoButton and websiteVideoButton to buttonPanel1, and add appButton and websiteButton to buttonPanel2.
15. Add buttonPanel1 and buttonPanel2 to the JFrame.
16. Resize "login\_image.jpg" to 65% of its original size to create an ImageIcon for the loginButton. Add an ActionListener to open a new instance of SoilEnthusiast when clicked.
17. Set the size of the JFrame to 1280x720 pixels.
18. Set the location of the JFrame to the center of the screen and make it visible.

File Name: SoilEnthusiast.java

1. **Define SoilEnthusiast Class**:
   * Define a class named **SoilEnthusiast**.
   * Implement a constructor for the class to perform certain actions when an instance is created.
2. **Constructor Actions**:
   * Inside the constructor, perform the following actions:
     + Execute the **runCommand** method with arguments to compile and run **DatabaseSetup.java**.
     + Execute the **runCommand** method with arguments to compile and run **TableSetup.java**.
     + Execute the **runCommand** method with arguments to compile and run **UserRegistrationForm.java**.
3. **Run Command Method**:
   * Define a method named **runCommand** that takes a variable number of String arguments representing a command to be executed.
   * Inside the **runCommand** method, create a **ProcessBuilder** instance with the provided command arguments.
   * Try to start the process using **process.waitFor()** to wait for the process to complete.
   * Handle **IOException** and **InterruptedException** if they occur during process execution.
4. **Main Method**:
   * Define the **main** method.
   * Inside the **main** method, create an instance of **SoilEnthusiast** to run the commands specified in its constructor.
5. **Execution Flow**:
   * When the program is executed, it creates an instance of **SoilEnthusiast**.
   * The constructor of **SoilEnthusiast** is invoked, which in turn executes the specified commands using the **runCommand** method.
   * Each command compiles and runs a Java file (**DatabaseSetup.java**, **TableSetup.java**, **UserRegistrationForm.java**) using the command line.

File Name: DatabaseSetup.java

1. **Import Required Libraries**:
   * Import necessary Java libraries for JDBC (Java Database Connectivity) and logging.
2. **Define DatabaseSetup Class**:
   * Define a class named **DatabaseSetup**.
3. **Class Variables**:
   * Define class variables for JDBC connection URL, username, password, and a Logger instance for logging.
4. **Main Method**:
   * Define the **main** method as the entry point of the class.
   * Inside the **main** method, establish a connection to the MySQL database using **DriverManager.getConnection**.
5. **Database Setup**:
   * Inside the try-with-resources block for the connection, create a **PreparedStatement** to execute SQL commands.
   * Use the **PreparedStatement** to execute an SQL command to create the database **soil\_enthusiast** if it does not already exist (**CREATE DATABASE IF NOT EXISTS soil\_enthusiast**).
6. **Exception Handling**:
   * Catch **SQLException** if any database-related errors occur during execution.
   * Use the logger to log the error message and stack trace at the **SEVERE** level.
7. **Execution Flow**:
   * When the program is executed, it attempts to establish a connection to the MySQL database using the specified URL, username, and password.
   * If the connection is successful, it executes an SQL command to create the database **soil\_enthusiast** if it doesn't already exist.
   * If any errors occur during database setup, they are caught and logged using the logger.

File Name: TableSetup.java

1. **Import Required Libraries**:
   * Import necessary Java libraries for JDBC (Java Database Connectivity) and logging.
2. **Define TableSetup Class**:
   * Define a class named **TableSetup**.
3. **Class Variables**:
   * Define class variables for JDBC connection URL, username, password, and a Logger instance for logging.
   * Define SQL statements to create the required tables in the database.
4. **Main Method**:
   * Define the **main** method as the entry point of the class.
   * Inside the **main** method, establish a connection to the MySQL database using **DriverManager.getConnection**.
5. **Create Tables**:
   * Inside the try-with-resources block for the connection, call the **createTable** method for each SQL statement to create the required tables.
   * The **createTable** method takes a connection and an SQL statement as parameters and executes the statement to create the table.
6. **Exception Handling**:
   * Catch **SQLException** if any database-related errors occur during table creation.
   * Use the logger to log the error message and stack trace at the **SEVERE** level.
7. **Execution Flow**:
   * When the program is executed, it attempts to establish a connection to the MySQL database using the specified URL, username, and password.
   * If the connection is successful, it executes SQL statements to create the required tables (**login**, **register**, **order\_details**, **CompostCalculated**) if they don't already exist.
   * If any errors occur during table creation, they are caught and logged using the logger.

File Name: UserRegistrationForm.java

1. **Import Required Libraries**:
   * Import necessary Java libraries for Swing components, event handling, and JDBC (Java Database Connectivity).
2. **Define UserRegistrationForm Class**:
   * Define a class named **UserRegistrationForm** that extends **JFrame** and implements the **ActionListener** interface.
3. **Instance Variables**:
   * Declare instance variables for the Swing components needed in the registration form, such as text fields, password field, buttons, and labels.
4. **Constructor**:
   * Implement the constructor for the registration form.
   * Set the title, default close operation, size, location, and layout (using **GridLayout**) for the frame.
   * Initialize all the Swing components and add them to the frame.
5. **Main Method**:
   * Define the **main** method to create an instance of **UserRegistrationForm** using **SwingUtilities.invokeLater**.
6. **ActionListener Implementation**:
   * Implement the **actionPerformed** method to handle button clicks and user registration logic.
   * Check which button is clicked using **e.getSource()** and perform the corresponding action.
   * If the register button is clicked:
     + Retrieve input values from the text fields and password field.
     + Validate the email format using a regular expression.
     + Establish a connection to the MySQL database using JDBC.
     + Prepare and execute SQL statements to insert user data into the **register** and **login** tables.
     + Display success or error messages using **JOptionPane**.
     + Clear the form fields after registration.
   * If the login button is clicked:
     + Open a new login frame (assuming the **LoginFrame** class exists).
     + Dispose of the registration frame to switch to the login frame.
7. **isValidEmail Method**:
   * Implement a method named **isValidEmail** to perform basic email validation using a regular expression.
8. **Execution Flow**:
   * When the program is executed, it creates an instance of **UserRegistrationForm** with the registration form GUI.
   * The user can input registration details, click the register button to store data in the database, or click the login button to switch to the login frame.
   * Input validation and database operations are performed based on user actions.

File Name: LoginFrame.java

1. **Import Required Libraries**:
   * Import necessary Java libraries for Swing components, event handling, and JDBC (Java Database Connectivity).
2. **Define LoginFrame Class**:
   * Define a class named **LoginFrame** that extends **JFrame** and implements the **ActionListener** interface.
3. **Instance Variables**:
   * Declare instance variables for the Swing components needed in the login frame, such as text fields, password field, and buttons.
4. **Constructor**:
   * Implement the constructor for the login frame.
   * Set the title, default close operation, size, and location for the frame.
   * Create a JPanel with a grid layout to organize the components.
   * Initialize and add the username label, username field, password label, password field, login button, and sign-up button to the panel.
   * Add the panel to the frame and set it visible.
5. **Main Method**:
   * Define the **main** method to create an instance of **LoginFrame** using **SwingUtilities.invokeLater**.
6. **ActionListener Implementation**:
   * Implement the **actionPerformed** method to handle button clicks and login/sign-up logic.
   * Check which button is clicked using **e.getSource()** and perform the corresponding action.
   * If the login button is clicked:
     + Retrieve the entered username and password.
     + Establish a connection to the MySQL database using JDBC.
     + Prepare and execute an SQL query to check if the username and password match in the **login** table.
     + If a match is found, show a success message and open the user dashboard frame (**UserDashboard**) passing the username.
     + Dispose of the login frame.
     + If no match is found, show an error message.
   * If the sign-up button is clicked:
     + Open a new user registration form (**UserRegistrationForm**).
     + Dispose of the login frame.
7. **Execution Flow**:
   * When the program is executed, it creates an instance of **LoginFrame** with the login form GUI.
   * The user can input login details, click the login button to authenticate, or click the sign-up button to register as a new user.
   * Input validation, database queries, and frame switching are performed based on user actions.

File Name: UserDashboard.java

1. **Import Required Libraries**:
   * Import necessary Java libraries for Swing components, event handling, image handling, and data structures.
2. **Define UserDashboard Class**:
   * Define a class named **UserDashboard** that extends **JFrame**.
   * Include instance variables for components like text areas, labels, buttons, total amount, product list, and username.
3. **Constructor**:
   * Implement the constructor to initialize the user dashboard.
   * Set the title, size, and layout of the frame.
   * Create panels for header, content, products, cart, and bottom buttons.
   * Add labels, combo box, scroll pane, buttons, and action listeners for various functionalities like category selection, adding/removing items to/from cart, resetting cart, checkout, logout, and compost calculator.
4. **Product Class**:
   * Define an inner class named **Product** to represent each product with attributes like name, image path, price, amount, and category.
5. **Update Product Panel**:
   * Implement the **updateProductPanel** method to refresh the products displayed based on the selected category.
   * Iterate over the product list and add product panels to the products panel if they match the selected category.
6. **Add Product Panel**:
   * Implement the **addProductPanel** method to create and add a panel for each product.
   * Include image handling using **ImageIO** to display product images.
   * Add buttons for adding/removing items to/from the cart with corresponding action listeners to update the cart and total amount.
7. **Reset Cart**:
   * Implement the **resetCart** method to clear the cart text area and reset the total amount to zero.
8. **Show Billing Dashboard**:
   * Implement the **showBillingDashboard** method to create and display the billing dashboard when the checkout button is clicked.
9. **Main Method**:
   * Define the **main** method to create an instance of **UserDashboard** with a sample username and make it visible.
10. **Execution Flow**:
    * When the program is executed, it creates an instance of **UserDashboard** with the user dashboard GUI.
    * Users can browse through different categories of eco-friendly products, add items to the cart, remove items from the cart, reset the cart, checkout, logout, and access the compost calculator.
    * Product information is displayed dynamically based on the selected category.
    * Cart updates and total amount calculations are performed based on user interactions with the buttons.

File Name: CompostCalculatorFrame.java

1. **Import Required Libraries**:
   * Import necessary Java libraries for Swing components, event handling, and database connectivity.
2. **Define CompostCalculatorFrame Class**:
   * Define a class named **CompostCalculatorFrame** that extends **JFrame**.
   * Include instance variables for components like labels, combo box, text fields, buttons, and result area.
   * Also, include a variable to store the username and the compost needed.
3. **Constructor**:
   * Implement the constructor to initialize the Compost Calculator frame.
   * Set the title, default close operation, layout, and visibility of the frame.
   * Create and add components such as project type combo box, text fields, calculate button, result area, and scroll pane.
   * Set default values for length, width, and depth based on inputs from the previous frame (if any).
   * Automatically calculate the compost needed upon frame initialization.
4. **Calculate Compost Method**:
   * Implement the **calculateCompost** method to calculate the compost needed based on project type, length, width, and depth.
   * Use switch cases to handle different project types and calculate the compost accordingly.
   * Update the result area with the calculated compost needed in cubic yards.
   * Save the calculated compost data to the database if the compost needed is greater than zero.
5. **Save to Database Method**:
   * Implement the **saveToDatabase** method to save the compost calculation data to the database.
   * Use JDBC to establish a connection to the MySQL database.
   * Prepare and execute an SQL INSERT statement to insert the compost calculation data into the database table **CompostCalculated**.
   * Catch and handle any SQL exceptions that may occur during database operations.
6. **Main Method**:
   * No main method is included in this class as it is meant to be used as a part of a larger application where the compost calculator frame is opened from another frame.

File Name: BillingDashboard.java

1. **Import Required Libraries**:
   * Import necessary Java libraries for Swing components and UI-related operations.
2. **Define BillingDashboard Class**:
   * Define a class named **BillingDashboard** that extends **JFrame**.
   * Include instance variables for components like **JTextArea** for cart items, **JLabel** for total amount, and the username.
3. **Constructor**:
   * Implement the constructor to initialize the Billing Dashboard frame.
   * Set the title, default close operation, size, and location of the frame.
   * Create and add components such as title label, cart items text area, total amount label, and proceed to payment button.
4. **Proceed to Payment Button ActionListener**:
   * Add an action listener to the "Proceed to Payment" button.
   * When the button is clicked, display a message dialog to prompt the user to select a payment option.
   * Open a new **PaymentOptionsFrame** passing the product list, total amount, and username.
   * Close the current **BillingDashboard** frame.
5. **Main Method**:
   * Include a main method that creates an instance of **BillingDashboard**.
   * Pass sample product data, total amount, and a username to the constructor to demonstrate the functionality.

File Name: PaymentOptionsFrame.java

1. **Import Required Libraries**:
   * Import necessary Java libraries for Swing components, UI-related operations, file handling, and database connectivity.
2. **Define PaymentOptionsFrame Class**:
   * Define a class named **PaymentOptionsFrame** that extends **JFrame**.
   * Include instance variables for components like **JTextField** for address input, **JLabel** for address label, **String** for username, and a **JPanel** for buttons.
3. **Constructor**:
   * Implement the constructor to initialize the Payment Options frame.
   * Set the title, default close operation, size, and location of the frame.
   * Create and add components such as payment option radio buttons, address input field, and a "Proceed" button.
4. **Proceed Button ActionListener**:
   * Add an action listener to the "Proceed" button.
   * When the button is clicked, validate the selected payment option and address input.
   * Save order details to the database using the **saveOrderDetails** method.
   * Show success messages, generate a receipt using the **generateReceipt** method, and dispose all frames to open the home screen.
5. **Database Connection and Saving Order Details**:
   * Implement the **saveOrderDetails** method to connect to the database and insert order details (username, products, total amount, payment mode, and address) into the **order\_details** table.
6. **Generating Receipt**:
   * Implement the **generateReceipt** method to create a receipt file using a **BufferedWriter** and save it based on user choice using **JFileChooser**.
7. **Dispose All Frames and Open Home Screen**:
   * Implement the **disposeAllAndOpenHome** method to dispose all frames currently open and open the home screen.
8. **Main Method**:
   * Include a main method that creates an instance of **PaymentOptionsFrame** with sample data for demonstration purposes.