Nama: Rifki Abiyan

NIM: 2309106030

Kelas: A2 2023

Pemrograman Berbasis Objek

POSTTEST 5

WatchOperations.JAVA

```java
package com.precisionwatchcare;

public interface WatchOperations {
    void displayWatchDetails();
    void performMaintenance();
}
```

App.JAVA

App.java U

src > com > precisionwatchcare > App.java > ...

```java
public class App {
    public static void main(String[] args) {
        // admin acc = admin | admin123
        while (true) {
            System.out.println("\n=== Precision Watch Care ===");
            System.out.println("1. Register");
            System.out.println("2. Login");
            System.out.println("3. Exit");
            System.out.println("\n============================");
            System.out.print("Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    register();
                    break;
                case 2:
                    login();
                    break;
                case 3:
                    System.out.println("Exiting program...");
                    return;
                default:
                    System.out.println("Invalid option. Please try again.");
            }
        }
    }

```

```java
public class App {

    // Method Register
    private static void register() {
        System.out.print(s:"Enter username: ");
        String username = scanner.nextLine();
        System.out.print(s:"Enter password: ");
        String password = scanner.nextLine();

        for (UserAccount account : userAccounts) {
            if (account.getUsername().equals(username)) {
                System.out.println(x:"Username already exists. Please choose another username.");
                return;
            }
        }

        userAccounts.add(new UserAccount(username, password, role:"user"));
        System.out.println(x:"Registration successful! You can now login.");
    }

    // Method Login
    private static void login() {
        System.out.print(s:"Enter username: ");
        String username = scanner.nextLine();
        System.out.print(s:"Enter password: ");
        String password = scanner.nextLine();

        for (UserAccount account : userAccounts) {
            if (account.getUsername().equals(username) && account.getPassword().equals(password)) {
                System.out.println("Login successful! Welcome, " + username + ".");
                if (account.getRole().equals(anObject:"admin")) {
                    Admin.adminMenu();
                } else {
                    User.userMenu();
                }
                return;
            }
        }

        System.out.println(x:"Invalid username or password. Please try again.");
    }
}
```

Admin.java

```java
package com.precisionwatchcare;

import java.util.ArrayList;
import java.util.Scanner;
// admin acc = admin | admin123
public class Admin {
    private static Scanner scanner = new Scanner(System.in);

    public static void adminMenu() {
        while (true) {
            System.out.println(x:"\n===== Admin Menu =====");
            System.out.println(x:"1. View All Services");
            System.out.println(x:"2. Estimate Service Cost");
            System.out.println(x:"3. Update Service Status");
            System.out.println(x:"4. Return Watch");
            System.out.println(x:"5. Logout");
            System.out.println(x:"\n======================");
            System.out.print(s:"Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    viewAllServices();
                    break;
                case 2:
                    estimateServiceCost();
                    break;
                case 3:
                    updateServiceStatus();
```

```
                case 4:
                    returnWatch();
                    break;
                case 5:
                    System.out.println(x:"Logging out...");
                    return;
                default:
                    System.out.println(x:"Invalid option. Please try again.");
            }
        }
    }

    // Method Service
    private static void viewAllServices() {
        ArrayList<Service> serviceList = User.getServiceList();
        if (serviceList.isEmpty()) {
            System.out.println(x:"No services available.");
        } else {
            for (Service service : serviceList) {
                System.out.println(service);
            }
        }
    }

    // Method Service Cost
    private static void estimateServiceCost() {
        try {
            viewAllServices();
            if (!User.getServiceList().isEmpty()) {
                System.out.print(s:"Enter the service ID to estimate cost: ");
                String serviceId = scanner.nextLine();
                if (serviceId.isEmpty()) {
                    throw new IllegalArgumentException(s:"Service ID cannot be empty");
                }

                System.out.print(s:"Enter estimated cost: Rp");
                double cost = scanner.nextDouble();
                scanner.nextLine();
                if (cost <= 0) {
                    throw new IllegalArgumentException(s:"Cost must be positive");
                }

                boolean found = false;
                for (Service service : User.getServiceList()) {
                    if (service.getServiceId().equals(serviceId)) {
                        service.setCost(cost);
                        System.out.println(x:"Cost estimated successfully!");
                        found = true;
                        break;
                    }
                }
                if (!found) {
                    System.out.println(x:"Service ID not found.");
                }
            }
```

```
        } catch (InputMismatchException e) {
            System.out.println(x:"Error: Invalid input for cost. Please enter a number.");
            scanner.nextLine(); // Clear the invalid input
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("An unexpected error occurred: " + e.getMessage());
        }
    }

    // Method Status Update
    private static void updateServiceStatus() {
        viewAllServices();
        if (!User.getServiceList().isEmpty()) {
            System.out.print(s:"Enter the service ID to update status: ");
            String serviceId = scanner.nextLine();
            System.out.print(s:"Enter new status (Pending/In Progress/Completed): ");
            String status = scanner.nextLine();

            for (Service service : User.getServiceList()) {
                if (service.getServiceId().equals(serviceId)) {
                    service.setStatus(status);
                    System.out.println(x:"Status updated successfully!");
                    return;
                }
            }
            System.out.println(x:"Service ID not found.");
        }
    }

    // Method Return Watch
    private static void returnWatch() {
        viewAllServices();
        if (!User.getServiceList().isEmpty()) {
            System.out.print(s:"Enter the service ID to return watch: ");
            String serviceId = scanner.nextLine();

            for (Service service : User.getServiceList()) {
                if (service.getServiceId().equals(serviceId)) {
                    if (service.getStatus().equals(anObject:"Completed")) {
                        System.out.println(x:"Watch returned successfully!");
                        User.getServiceList().remove(service);
                    } else {
                        System.out.println(x:"Service is not yet completed.");
                    }
                    return;
                }
            }
            System.out.println(x:"Service ID not found.");
        }
    }
}
```

Service.java

```java
package com.precisionwatchcare;

public class Service {
    private String serviceId;
    private String serviceType;
    private double cost;
    private String status;
    private boolean useCourier;

    public Service(String serviceId, String serviceType, double cost, boolean useCourier) {
        this.serviceId = serviceId;
        this.serviceType = serviceType;
        this.cost = cost;
        this.status = "Pending"; // Default
        this.useCourier = useCourier;
    }

    // Overloaded constructor - without cost parameter (default to 0)
    public Service(String serviceId, String serviceType, boolean useCourier) {
        this(serviceId, serviceType, cost:0, useCourier);
    }

    // Overloaded constructor - without courier parameter (default to false)
    public Service(String serviceId, String serviceType) {
        this(serviceId, serviceType, cost:0, useCourier:false);
    }
```

```java

    public String getServiceId() {
        return serviceId;
    }

    public void setServiceId(String serviceId) {
        this.serviceId = serviceId;
    }

    public String getServiceType() {
        return serviceType;
    }

    public void setServiceType(String serviceType) {
        this.serviceType = serviceType;
    }

    public double getCost() {
        return cost;
    }

    public void setCost(double cost) {
        this.cost = cost;
    }

    public String getStatus() {
        return status;
    }
```

```
38
39      public String getServiceType() {
40          return serviceType;
41      }
42
43      public void setServiceType(String serviceType) {
44          this.serviceType = serviceType;
45      }
46
47      public double getCost() {
48          return cost;
49      }
50
51      public void setCost(double cost) {
52          this.cost = cost;
53      }
54
55      public String getStatus() {
56          return status;
57      }
58
59      public void setStatus(String status) {
60          this.status = status;
61      }
```

```
63      public boolean isUseCourier() {
64          return useCourier;
65      }
66
67      public void setUseCourier(boolean useCourier) {
68          this.useCourier = useCourier;
69      }
70
71      @Override
72      public String toString() {
73          return "Service [ID: " + serviceId + ", Type: " + serviceType +
74                  ", Cost: Rp" + cost + ", Status: " + status +
75                  ", Use Courier: " + useCourier + "]";
76      }
77  }
```

User.java

```java
package com.precisionwatchcare;

import java.util.ArrayList;
import java.util.Scanner;

public class User {
    private static ArrayList<Watch> watchList = new ArrayList<>();
    private static ArrayList<Service> serviceList = new ArrayList<>();
    private static Scanner scanner = new Scanner(System.in);

    public static void userMenu() {
        while (true) {
            System.out.println(x:"\n=== User Menu ===");
            System.out.println(x:"1. Add Watch");
            System.out.println(x:"2. View My Watches");
            System.out.println(x:"3. Request Service");
            System.out.println(x:"4. Logout");
            System.out.println(x:"\n================");
            System.out.print(s:"Choose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    addWatch();
                    break;
                case 2:
                    viewMyWatches();
                    break;
```

```java
                case 3:
                    requestService();
                    break;
                case 4:
                    System.out.println(x:"Logging out...");
                    return;
                default:
                    System.out.println(x:"Invalid option. Please try again.");
            }
        }
    }

    // Method Add Watch
    private static void addWatch() {
        try {
            System.out.print(s:"Enter brand: ");
            String brand = scanner.nextLine();
            if (brand.isEmpty()) {
                throw new IllegalArgumentException(s:"Brand cannot be empty");
            }

            System.out.print(s:"Enter model: ");
            String model = scanner.nextLine();
            if (model.isEmpty()) {
                throw new IllegalArgumentException(s:"Model cannot be empty");
            }

            System.out.print(s:"Enter year: ");
```

```
            int year = scanner.nextInt();
            scanner.nextLine();
            if (year < 1900 || year > java.time.Year.now().getValue()) {
                throw new IllegalArgumentException(s:"Invalid year");
            }

            System.out.print(s:"Enter complication/features: ");
            String complication = scanner.nextLine();
            System.out.print(s:"Enter type (Analog/Digital): ");
            String type = scanner.nextLine();

            Watch watch;
            if (type.equalsIgnoreCase(anotherString:"Analog")) {
                watch = new AnalogWatch(brand, model, year, complication);
            } else if (type.equalsIgnoreCase(anotherString:"Digital")) {
                watch = new DigitalWatch(brand, model, year, complication);
            } else {
                System.out.println(x:"Invalid type. Defaulting to Analog.");
                watch = new AnalogWatch(brand, model, year, complication);
            }

            watchList.add(watch);
            watch.displaySpecialFeature();
            watch.displayWatchDetails(); // Using interface method
            watch.performMaintenance(); // Using interface method
            System.out.println("Total watches in system: " + Watch.getTotalWatches()); // Using static method
            System.out.println(x:"Watch added successfully!");
        } catch (IllegalArgumentException e) {
```

```
            System.out.println("Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("An unexpected error occurred: " + e.getMessage());
        }
    }

    // Method View Watch
    private static void viewMyWatches() {
        if (watchList.isEmpty()) {
            System.out.println(x:"No watches available.");
        } else {
            for (Watch watch : watchList) {
                System.out.println(watch);
            }
        }
    }

    private static void requestService() {
        viewMyWatches();
        if (!watchList.isEmpty()) {
            System.out.print(s:"Enter the index of the watch to service: ");
            int index = scanner.nextInt();
            scanner.nextLine();

            if (index >= 0 && index < watchList.size()) {
                System.out.print(s:"Enter service type (Cleaning/Repair/Battery Change/Custom): ");
                String serviceType = scanner.nextLine();
```

```
                Service service;
                System.out.print(s:"Use courier? (y/n): ");
                String courierChoice = scanner.nextLine();

                if (courierChoice.equalsIgnoreCase(anotherString:"y")) {
                    service = new Service("SERV" + (serviceList.size() + 1), serviceType, useCourier:true);
                } else {
                    service = new Service("SERV" + (serviceList.size() + 1), serviceType);
                }

                serviceList.add(service);
                System.out.println("Service requested successfully! Service ID: " + service.getServiceId());
            } else {
                System.out.println(x:"Invalid index.");
            }
        }
    }

    public static ArrayList<Service> getServiceList() {
        return serviceList;
    }
}
```

UserAccount.java

```java
package com.precisionwatchcare;
public final class UserAccount {  // Made class final
    private final String username;
    private String password;
    private String role;

    public UserAccount(String username, String password, String role) {
        this.username = username;
        this.password = password;
        this.role = role;
    }

    // Getter dan Setter
    public String getUsername() {
        return username;
    }

    public String getPassword() {
        return password;
    }

    public String getRole() {
        return role;
    }

    @Override
    public String toString() {
        return "UserAccount [Username: " + username + ", Role: " + role + "]";
    }
}
```

Watch.java

```java
package com.precisionwatchcare;

public abstract class Watch implements WatchOperations {
    private String brand;
    protected String model;
    final int year;
    public String complication;
    private String type;
    private static int totalWatches = 0;

    public Watch(String brand, String model, int year, String complication, String type) {
        this.brand = brand;
        this.model = model;
        this.year = year;
        this.complication = complication;
        this.type = type;
        totalWatches++;
    }

    // Static method to get total watches
    public static int getTotalWatches() {
        return totalWatches;
    }

    // Implement interface methods
    @Override
    public void displayWatchDetails() {
        System.out.println("Brand: " + brand + ", Model: " + model + ", Year: " + year);
    }
```

```java
    @Override
    public void performMaintenance() {
        System.out.println("Performing basic maintenance for " + brand + " " + model);
    }

    // Added abstract method
    public abstract void displaySpecialFeature();

    // Getter dan Setter
    public final String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public String getModel() {
        return model;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public String getType() {
        return type;
    }
}
```

```java
    public void setType(String type) {
        this.type = type;
    }

    public String getComplication() {
        return complication;
    }

    @Override
    public String toString() {
        displaySpecialFeature();
        if (type.equalsIgnoreCase(anotherString:"Analog")) {
            return "Analog Watch [Brand: " + brand + ", Model: " + model +
                   ", Year: " + year + ", Complication: " + complication + "]";
        } else if (type.equalsIgnoreCase(anotherString:"Digital")) {
            return "Digital Watch [Brand: " + brand + ", Model: " + model +
                   ", Year: " + year + ", Features: " + complication + "]";
        }
        return "Watch [Brand: " + brand + ", Model: " + model +
               ", Year: " + year + ", Complication: " + complication + ", Type: " + type + "]";
    }
}
```

AnalogWatch.java

---

src > com > precisionwatchcare > AnalogWatch.java > AnalogWatch

```java
package com.precisionwatchcare;

public class AnalogWatch extends Watch {
    public AnalogWatch(String brand, String model, int year, String complication) {
        super(brand, model, year, complication, type:"Analog");
    }

    @Override
    public void displaySpecialFeature() {
        System.out.println("Special Feature: Mechanical movement with " + getComplication() + " complication");
    }
}
```

DigitalWatch.java

```java
package com.precisionwatchcare;

public class DigitalWatch extends Watch {
    public DigitalWatch(String brand, String model, int year, String features) {
        super(brand, model, year, features, type:"Digital");
    }

    @Override
    public void displaySpecialFeature() {
        System.out.println("Special Feature: Electronic display with " + getComplication() + " features");
    }
}
```

Cara menjalankan program