Advance Courier Service

# Team Details:-

1. HARSH DEVRE [TEAM LEADER]
2. VRAJ SHAH [MEMBERS]
3. DEV PANCHAL [MEMBERS]
4. VISHVJIT PATEL [MEMBERS]

\*\*Project Title:\*\* Advance Courier Service Management System

Project Description:-

The \*\*Advance Courier Service Management System\*\* is a Java application designed to simplify and enhance the management of courier services by providing an intuitive, interactive console-based interface. This system allows for efficient tracking and updating of courier packages, enabling real-time management of couriers, their statuses, and client details.

The project is structured around two main classes: `Courier` and `CourierService`, alongside a main application class, `Main`, which drives the user interaction. Here’s a breakdown of each component:

### Key Components

1. Courier Class:

- The `Courier` class represents an individual courier package with specific attributes:

- Tracking Number: Unique identifier for each package.

- Sender: Name of the person or entity sending the package.

- Recipient: Name of the person or entity receiving the package.

- Status: The current status of the package, defaulting to "In Transit" upon creation.

- The class provides getter methods for retrieving each attribute and a method to update the package's status.

- The `toString()` method overrides the default object description to output a readable string representing the package details.

2. CourierService Class:-

- The `CourierService` class manages a collection of `Courier` objects, stored in an `ArrayList`.

- This class includes the following key methods:

- addCourier: Adds a new courier to the list.

- findCourier: Searches for a courier by its tracking number.

- updateCourierStatus: Finds a specific courier by its tracking number and updates its status if found.

- listAllCouriers: Lists all couriers currently managed by the system.

- Error handling is implemented to inform users if a specified courier cannot be found.

3. Main Application (Main Class):

- The `Main` class serves as the user interface, displaying options for interacting with the system and reading user input through the `Scanner` class.

- It provides the following interactive options:

- Add Courier: Prompts the user to input the tracking number, sender, and recipient, then adds the courier to the system.

- Update Courier Status: Requests a tracking number and new status, then updates the corresponding courier’s status.

- List All Couriers: Lists details of all couriers currently in the system.

- Find Courier by Tracking Number: Allows the user to search for a specific courier package by tracking number.

- Exit: Closes the application.

- Input validation is implemented for smooth user interaction, catching invalid inputs and providing prompts to re-enter information when necessary.

### Key Features

- Easy Courier Management: With this system, users can add, update, and track couriers efficiently, ensuring a streamlined process for small to medium-sized courier services.

- Real-Time Updates: Statuses of couriers can be modified on the go, reflecting real-time package tracking.

- Error Handling:Input validation and error messages help users navigate the system without frustration.

### Usage Scenarios

This project would be especially useful for small businesses or startup courier services looking to automate the management of shipments without the need for complex or expensive software solutions. The straightforward interface and functionality allow users to easily track and update packages, ensuring both sender and recipient have up-to-date information on delivery statuses.

=>Team Members:

- **HARSH DEVRE:** Lead Developer, responsible for core functionality and integration.

- VRAJ SHAH: QA Tester, Focused on Testing the Code flow.

- DEV PANCHAL:QA Tester, implemented testing and validation checks for all functionalities.

- VISHVJIT PATEL: Project Manager, ensured timely delivery of each component and maintained the project roadmap.

This project was built with simplicity, usability, and reliability in mind, making it an ideal solution for managing couriers efficiently.

PROGRAM :-

AIM- MP2 Develop a Product on Advance Courier Service

**CODE-**

|  |
| --- |
| **import java.util.ArrayList;**  **import java.util.Scanner;**  **import java.util.InputMismatchException;**  **class Courier {**  **private String trackingNumber;**  **private String sender;**  **private String recipient;**  **private String status;**  **public Courier(String trackingNumber, String sender, String recipient) {**  **this.trackingNumber = trackingNumber;**  **this.sender = sender;**  **this.recipient = recipient;**  **this.status = "In Transit";**  **}**  **public String getTrackingNumber() {**  **return trackingNumber;**  **}**  **public String getSender() {**  **return sender;**  **}**  **public String getRecipient() {**  **return recipient;**  **}**  **public String getStatus() {**  **return status;**  **}**  **public void updateStatus(String status) {**  **this.status = status;**  **}**  **@Override**  **public String toString() {**  **return "Tracking Number: " + trackingNumber + ", Sender: " + sender + ", Recipient: " + recipient + ", Status: " + status;**  **}**  **}**  **class CourierService {**  **private ArrayList<Courier> couriers;**  **public CourierService() {**  **couriers = new ArrayList<>();**  **}**  **public void addCourier(Courier courier) {**  **couriers.add(courier);**  **}**  **public Courier findCourier(String trackingNumber) {**  **for (Courier courier : couriers) {**  **if (courier.getTrackingNumber().equals(trackingNumber)) {**  **return courier;**  **}**  **}**  **return null;**  **}**  **public void updateCourierStatus(String trackingNumber, String status) {**  **Courier courier = findCourier(trackingNumber);**  **if (courier != null) {**  **courier.updateStatus(status);**  **} else {**  **System.out.println("Courier not found.");**  **}**  **}**  **public void listAllCouriers() {**  **for (Courier courier : couriers) {**  **System.out.println(courier);**  **}**  **}**  **}**  **public class Main {**  **public static void main(String[] args) {**  **System.out.println("Welcome to Advance Courier Service");**  **Scanner scanner = new Scanner(System.in);**  **CourierService courierService = new CourierService();**  **while (true) {**  **System.out.println("\n1. Add Courier");**  **System.out.println("2. Update Courier Status");**  **System.out.println("3. List All Couriers");**  **System.out.println("4. Find Courier by Tracking Number");**  **System.out.println("5. Exit");**  **System.out.print("Choose an option: ");**  **int choice = 0;**  **try {**  **choice = scanner.nextInt();**  **} catch (InputMismatchException e) {**  **System.out.println("Invalid input. Please enter a number.");**  **scanner.next(); // Clear invalid input**  **continue;**  **}**  **scanner.nextLine(); // Consume newline**  **switch (choice) {**  **case 1:**  **System.out.print("Enter Tracking Number: ");**  **String trackingNumber = scanner.nextLine();**  **System.out.print("Enter Sender: ");**  **String sender = scanner.nextLine();**  **System.out.print("Enter Recipient: ");**  **String recipient = scanner.nextLine();**  **Courier courier = new Courier(trackingNumber, sender, recipient);**  **courierService.addCourier(courier);**  **System.out.println("Courier added successfully.");**  **break;**  **case 2:**  **System.out.print("Enter Tracking Number: ");**  **trackingNumber = scanner.nextLine();**  **System.out.print("Enter New Status: ");**  **String status = scanner.nextLine();**  **courierService.updateCourierStatus(trackingNumber, status);**  **System.out.println("Courier status updated successfully.");**  **break;**  **case 3:**  **System.out.println("Listing all couriers:");**  **courierService.listAllCouriers();**  **break;**  **case 4:**  **System.out.print("Enter Tracking Number: ");**  **trackingNumber = scanner.nextLine();**  **Courier foundCourier = courierService.findCourier(trackingNumber);**  **if (foundCourier != null) {**  **System.out.println("Courier found: " + foundCourier);**  **} else {**  **System.out.println("Courier not found.");**  **}**  **break;**  **case 5:**  **System.out.println("Exiting...");**  **scanner.close();**  **return;**  **default:**  **System.out.println("Invalid choice. Please try again.");**  **}**  **}**  **}**  **}** |

**OUTPUT-**

|  |
| --- |
|  |

The Java code you've provided implements a simple console-based courier management system. This system allows users to manage couriers by adding new shipments, updating their status, listing all couriers, and searching for a courier by its tracking number. Below is a detailed explanation of the code, along with instructions on how to run it.

Code Explanation

1. **Courier Class**

This class represents a courier with the following attributes:

* trackingNumber: Unique identifier for each courier.
* sender: Name of the person or entity sending the package.
* recipient: Name of the person or entity receiving the package.
* status: Current status of the courier (default is "In Transit").

**Methods:**

* Constructor to initialize a new courier.
* Getters for accessing the attributes.
* updateStatus(String status): Updates the status of the courier.
* toString(): Returns a string representation of the courier's details.

2. **CourierService Class**

This class manages a list of couriers and provides methods to:

* addCourier(Courier courier): Adds a new courier to the list.
* findCourier(String trackingNumber): Searches for a courier by its tracking number.
* updateCourierStatus(String trackingNumber, String status): Updates the status of a specified courier.
* listAllCouriers(): Prints details of all couriers in the system.

3. **Main Class**

This is the entry point of the application where user interaction takes place:

* A menu is displayed with options to add a courier, update its status, list all couriers, find a courier by tracking number, or exit the program.
* Input is handled using Scanner, and exceptions are managed to ensure valid input.

How to Run the Code

Step 1: Set Up Your Java Environment

Make sure you have Java Development Kit (JDK) installed on your machine. You can download it from [Oracle's website](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html) or use a package manager.

Step 2: Create a New Java File

1. Open your preferred text editor or Integrated Development Environment (IDE) like IntelliJ IDEA or Eclipse.
2. Copy and paste the provided code into a new file named Main.java.

Step 3: Compile the Code

Open your command prompt (Windows) or terminal (Mac/Linux), and navigate to the directory where you saved Main.java. Use the following command to compile:

bash

javac Main.java

If there are no errors, this will create a Main.class file in the same directory.

Step 4: Run the Program

After successful compilation, run the program using:

bash

java Main

Step 5: Interact with the Application

Follow these steps in the console:

1. Choose an option from the menu by entering its corresponding number.
2. If adding a courier, provide required details such as tracking number, sender, and recipient.
3. For updating status or finding a courier, enter the tracking number as prompted.
4. To exit, select option 5.

Example Interaction

text

Welcome to Advance Courier Service

1. Add Courier

2. Update Courier Status

3. List All Couriers

4. Find Courier by Tracking Number

5. Exit

Choose an option: 1

Enter Tracking Number: ABC123

Enter Sender: John Doe

Enter Recipient: Jane Smith

Courier added successfully.

Choose an option: 3

Listing all couriers:

Tracking Number: ABC123, Sender: John Doe, Recipient: Jane Smith, Status: In Transit

Choose an option: 5

Exiting...

=>Conclusion:-

This simple console-based application provides essential functionalities for managing couriers effectively. It can be further enhanced by adding features such as data persistence (using databases), user authentication, and more sophisticated error handling for production-level applications.