# Courier Management System

## Coding Task 3

## Arrays and Data Structures

---

**7. Create an array to store the tracking history of a parcel, where each entry represents a location update.**

**8. Implement a method to find the nearest available courier for a new order using an array of couriers.**

---

Task 3 demonstrates the use of arrays and data structures in Java by implementing functionalities related to parcel tracking and courier assignment.

The following tasks are covered:

**Task 3.1: Update Tracking History of a Parcel**

**Task 3.2: Find Nearest Available Courier**

The implementation makes use of an ArrayList for storing tracking updates dynamically and an array for managing courier locations.

---

**Main Method (psvm)**

The main method serves as the entry point of the program, offering a menu-driven interface for users to choose between:

- Updating the tracking history of a parcel.
- Finding the nearest available courier.
- Exiting the program.

A Scanner object is used to take user input, and an ArrayList<String> is utilized for managing the tracking history dynamically.

```
package main;

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

public class Task3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<String> trackingHistory = new ArrayList<>();
```

```
    while (true) {
        System.out.println("\nTask 3 - Choose an option:");
        System.out.println("1. Update Tracking History of a Parcel");
        System.out.println("2. Find Nearest Available Courier");
        System.out.println("3. Exit");
        System.out.print("Enter your choice: ");

        int choice = scanner.nextInt();
        scanner.nextLine();

        switch (choice) {
            case 1:
                updateTrackingHistory(scanner, trackingHistory);
                break;
            case 2:
                findNearestCourier(scanner);
                break;
            case 3:
                System.out.println("Exiting Task 3...");
                scanner.close();
                return;
            default:
                System.out.println("Invalid choice! Please select a valid option.");
        }
    }
}
```

---

**Task 3.1: Update Tracking History of a Parcel**

**Create an array to store the tracking history of a parcel, where each entry represents a location update.**

This feature allows users to log tracking updates for a parcel, storing them in an ArrayList to enable dynamic data management.

**Code :**

```
// 1. Update Tracking History of a Parcel
public static void updateTrackingHistory(Scanner scanner, ArrayList<String> trackingHistory) {
    while (true) {
        System.out.print("Enter tracking update (or type 'exit' to finish): ");
        String update = scanner.nextLine().trim();

        if (update.equalsIgnoreCase( anotherString: "exit")) {
            break;
        }
        trackingHistory.add(update);
    }

    System.out.println("\nTracking History:");
    for (String record : trackingHistory) {
        System.out.print("--> " + record);
    }
}
```

**Output :**

```
Task 3 - Choose an option:
1. Update Tracking History of a Parcel
2. Find Nearest Available Courier
3. Exit
Enter your choice: 1
Enter tracking update (or type 'exit' to finish): Order placed
Enter tracking update (or type 'exit' to finish): Processing
Enter tracking update (or type 'exit' to finish): At warehouse
Enter tracking update (or type 'exit' to finish): In transit
Enter tracking update (or type 'exit' to finish): Shipment Ready
Enter tracking update (or type 'exit' to finish): Shipped
Enter tracking update (or type 'exit' to finish): Reached to nearesh hub
Enter tracking update (or type 'exit' to finish): out for delivery
Enter tracking update (or type 'exit' to finish): Delivery
Enter tracking update (or type 'exit' to finish): exit

Tracking History:
--> Order placed--> Processing--> At warehouse--> In transit--> Shipment Ready--> Shipped--> Reached to nearesh hub--> out for delivery--> Delivery
```

---

## Task 3.2: Find Nearest Available Courier

**Implement a method to find the nearest available courier for a new order using an array of couriers.**

This functionality calculates the absolute distance between a shipment location and the locations of couriers stored in an array to determine the nearest available courier.

**Code :**

```java
// 2. Find Nearest Available Courier
public static void findNearestCourier(Scanner scanner) {
    String[] couriers = {"Courier A", "Courier B", "Courier C", "Courier D"};
    int[] locations = {5, 10, 3, 8};

    System.out.print("Enter shipment location (distance in km): ");
    int shipmentLocation = scanner.nextInt();

    String nearestCourier = null;
    int minDistance = Integer.MAX_VALUE;

    for (int i = 0; i < couriers.length; i++) {
        int distance = Math.abs(locations[i] - shipmentLocation);
        if (distance < minDistance) {
            minDistance = distance;
            nearestCourier = couriers[i];
        }
    }

    System.out.println("Nearest available courier: " + nearestCourier + " (Distance: " + minDistance + " km)");
}
```

**Output :**

```
Task 3 - Choose an option:
1. Update Tracking History of a Parcel
2. Find Nearest Available Courier
3. Exit
Enter your choice: 2
Enter shipment location (distance in km): 5
Nearest available courier: Courier A (Distance: 0 km)
```

```
Enter shipment location (distance in km): 9
Nearest available courier: Courier B (Distance: 1 km)
```

```
Enter shipment location (distance in km): 7
Nearest available courier: Courier D (Distance: 1 km)
```

---

**Conclusion**

This task effectively demonstrates the use of arrays and ArrayList in Java for managing logistics operations.

- **Task 3.1** showcases dynamic storage of parcel tracking updates.
- **Task 3.2** highlights the efficiency of array-based search and comparison for courier selection.

These implementations provide a practical application of data structures in real-world delivery and tracking systems.

---