NileMart eCommerce Order Delivery System

Introduction

The NileMart Portal hosts a website for customers to search and select products, and place orders. They have a delivery system to ship orders from their warehouse to the customer's doorstep.

The project mimics an **order** delivery from warehouse to the **destination** city. Every destination has a fixed **delivery route**, for each **delivery type** (**normal** or **premium**), from the central warehouse.

There are several types of dispatch between centers:

- Truck
- Train
- Flight
- Boat (New)
- Ship (New)

The code under **v1** is fully implemented and works fine. We need your help in releasing a **v2** version with two improvements.

Housekeeping points

- This is a minimal example and may not follow some standard practices
- Print statements are added to help highlight the flow
- We focus on the main flow, and not any error handling. We expect you to enhance the basic flow without worrying about error conditions here.
- All classes and the main code is present in the same file, for ease of access - not the general practice.
- To avoid handling command-line arguments, config file paths are hard-coded in the source file - not a general practice.

Program Organization

Please look at the **v1** folder first

- The source file **nile-mart.py** in the **src/** directory, and multiple config files in the **config/** directory

- **config/order_batch.txt** contains one order per line in this format:
  <Order ID>-<Order Item>-<Customer-Name>-<Order Date>-<City>-<Delivery Date>-<Delivery Type>

- **config/normal_delivery_map.txt** and **config/premium_delivery_map.txt** hold routing information for each destination, per delivery type (**normal**/**premium**):
  <Destination> <Center>-<Dispatch Type>-<Center>,<Center>-<Dispatch Type>-<Center>,...,<Center>-<DispatchType>-<Destination>

- Classes
  - **OrderBatch** parses order information from **order_batch.txt** and creates order objects
  - **Order** stores details of an individual order, and triggers delivery via a route
  - **DeliveryMap** parses routing information and stores it in a raw format
  - **DeliverySystem** uses DeliveryMap to create usable routes to all destinations
  - **DeliveryRoute** class holds the structured route to a destination, with several stages
  - **DeliveryStage**, and the inherited classes -- **TrainDispatch**, **FlightDispatch**, **TruckDispatch** define specific stages of a route, depending on the dispatch type

- We have two delivery systems - **normal** and **premium**. We assign each order a route, based on delivery type, and destination city. The dispatch internally calls the route to process the order in a double dispatch manner.

- Key Highlights
  - DeliverySystem behaves as a **Builder**, converting route directions to a usable routing system
  - A **Strategy** pattern decides between using the normal or the premium route (to the same destination). It selects the routing scheme based on the delivery type mentioned in the order. The methods remain unchanged, but the order flows through a different route.
  - The stages in the route behave as a chain, calling the next in sequence, until all the stages are completed - implementing the **Chain of Responsibility** pattern
  - The route is passed to the order, which invokes order handling on the route. This double dispatch makes it more flexible to add say, digital deliveries and store pickups.

Problem Statement

We need two enhancements for version **v2**:

1. Only one **delivery_map.txt** file is used in **v2**. The format has changed to include both normal and premium types in the same file:

   <Destination> <Delivery Type> <Center>-<Dispatch Type>-<Center>,<Center>-<Dispatch Type>-<Center>,...,<Center>-<Dispatch Type>-<Destination>

   HINT: Note the two fields (**<Destination> and <Delivery Type>).** You can implement this field as a Python **tuple - (destination,delivery_type)** - in your delivery map data structure. Alternatively, you can implement it as a nested dictionary with delivery types within each destination.

   In **v1**, creating the delivery system is spread across:
   - The **DeliverySystem builder**
   - The client: creates different systems from separate config files, and chooses among them based on the delivery type.

   - All this should be combined in one **DeliverySystem**, to put the logic together, and provide a common endpoint to the client.

   - We have modified the config file. We need you to modify the code so that the **DeliverySystem** and other classes can use this config file, to create all types of routes to all destinations, in one go.

   - This should also make it easy to create and select a route, based on delivery type and destination city.

2. NileMart is expanding to nearby islands (**Tiliana**, **Marbut**). We need new methods of delivery, namely boat and ship. You would notice we added these two destinations, and two new delivery types in the config files. Please add code to include these new delivery methods.

Evaluation Rubric

Total Project Points: **240**

- Basic compilation without errors : **24 Points**
- Correctness:
  Correctness of output (The output is already being printed)
    - Problem statement - point 1 (**30%**) : **72 Points**
    - Problem statement - point 2 (**20%**) : **48 Points**
- Design:
  - Syntactic code smells (**5%**)
    - Consistency in naming : **12 Points**
  - Semantic Code Smells (**15%**)
    - No Code Duplication : **12 Points**
    - Function size, parameters, return values : **12 Points**
    - Class size, scope : **12 Points**
  - Design Structure (**20%**)
    - The builder should incorporate the overall route logic : **24 Points**
    - Use existing inheritance patterns for new dispatch types : **24 Points**

Program Instructions

1. Download the zipped folder named **M01-W01-04-Project-Assessment-Nile-Mart.zip**, unzip it on your local machine, and save it. Go into the directory named **M01-W01-04-Project-Assessment-Nile-Mart/.**

2. Make sure you have Python 3.6 or higher installed. At your command prompt, run:

   ```
   $ python --version
   Python 3.7.3
   ```

   If not installed, install the latest available version of Python 3.

3. To run the source code in either **v1** or **v2**, go to the **src/** folder and run the following command:

   ```
   $ python3 nile-mart.py (On many Linux platforms)
             OR
   $ python nile-mart.py  (On Windows platforms)
   ```

In any case, one of these two commands should work.

4. **Alternatively, you could install a popular Python IDE, such as PyCharm or Visual Studio Code, and select a command to build the project from there.**

The command will translate the source file into a Python 3 executable, and run it, showing you output in case of the **v1** version.

The source code under version **v2** is incomplete, and hence the command will fail to run the program. You need to solve our problem to make this file execute, giving the desired output.

5. **Output Format: Since we have given you a fully working version v1, you can enhance your version v2 to behave similarly to v1.**

   If there are doubts, or there is any ambiguity on the (exact) desired output format, make reasonable assumptions, state them explicitly, and implement them in your submissions.

   However, the final output should display all the required information, in its entirety. The assumptions should be restricted to the output format only.

6. Once the program under **v2** is ready to submit, zip the parent folder **M01-W01-04-Project-Assessment-Nile-Mart/**, and upload the new zip file as **M01-W01-04-Project-Assessment-Nile-Mart.zip**.It is now ready for evaluation.