

## Question 03: Drone Delivery System (LahoreLogistics)

**Goal:** Model a drone delivery system using Object-Oriented Programming (OOP) to simulate package assignments and status updates. **Topics:** OOP (Classes, Encapsulation, Composition, Aggregation).

### Tasks

#### 1. Entity Classes:

- Create a **Package class** (with `packageId`, `weightInKg` ).
- Create a **Drone class** (with `droneId`, `maxLoadInKg` ).

```
In [11]: class Package:  
    def __init__(self, packageId, weightInKg):  
        self.packageId = packageId  
        self.weightInKg = weightInKg
```

#### 1. Encapsulation:

- The Drone must protect its internal state by using a "**private**" `_status` **attribute**.
- This attribute should only be changeable via public methods: `getStatus()` and `setStatus(newStatus)` .
- The setter must validate that the `newStatus` is one of ('idle', 'delivering', 'charging') .

```
In [12]: class Drone:
    def __init__(self, droneId, maxLoadInKg):
        self.droneId = droneId
        self.maxLoadInKg = maxLoadInKg
        self._status = 'idle' # Private attribute
        self.current_package = None # Attribute to hold the assigned package

    def getStatus(self):
        return self._status

    def setStatus(self, newStatus):
        valid_statuses = ('idle', 'delivering', 'charging')
        if newStatus in valid_statuses:
            self._status = newStatus
        else:
            print(f"Invalid status: {newStatus}. Status must be one of {valid_statuses}")

    def assignPackage(self, packageObj):
        if self._status == 'idle' and packageObj.weightInKg <= self.maxLoadInKg:
            self.current_package = packageObj
            self.setStatus('delivering')
            print(f"Package {packageObj.packageId} assigned to Drone {self.droneId}")
            return True
        elif self._status != 'idle':
            print(f"Drone {self.droneId} is not idle. Cannot assign package.")
            return False
        else: # packageObj.weightInKg > self.maxLoadInKg
            print(f"Package {packageObj.packageId} exceeds Drone {self.droneId}'s max load.")
            return False
```

## 1. Aggregation:

- A Drone carries a Package for a temporary delivery, but the Package exists independently of the drone.
- Implement Drone.assignPackage(packageObj) .
- This method should only succeed if the drone's status is 'idle' AND the package's weight is **within the drone's maxLoadInKg** .

```
In [13]: # Create a Package and a Drone object
package1 = Package(packageId="P001", weightInKg=5)
drone1 = Drone(droneId="D001", maxLoadInKg=10)

# Attempt to assign the package to the drone
drone1.assignPackage(package1)

# Create a heavier package and another drone
package2 = Package(packageId="P002", weightInKg=15)
drone2 = Drone(droneId="D002", maxLoadInKg=10)

# Attempt to assign the heavier package
drone2.assignPackage(package2)

# Create an idle drone and a package
drone3 = Drone(droneId="D003", maxLoadInKg=20)
package3 = Package(packageId="P003", weightInKg=10)

# Assign the package to the idle drone
drone3.assignPackage(package3)

# Attempt to assign another package to the busy drone3
package4 = Package(packageId="P004", weightInKg=5)
drone3.assignPackage(package4)

# Display the state of the drones
print(drone1.getStatus())
print(drone2.getStatus())
print(drone3.getStatus())
```

```
Package P001 assigned to Drone D001
Package P002 exceeds Drone D002's max load.
Package P003 assigned to Drone D003
Drone D003 is not idle. Cannot assign package.
delivering
idle
delivering
```

## 1. Composition:

- A **FleetManager class** owns its fleet of **Drones**. If the manager is shut down, its **Drones** are part of that shutdown.
- The `__init__` method should create a dictionary to store and manage the lifecycle of its **Drone** objects.
- It should also have a list to track pending **Packages**.

```
In [14]: class FleetManager:
    def __init__(self):
        self.drones = {} # Composition: FleetManager owns Drone objects
        self.pending_packages = [] # Aggregation: FleetManager has a list of pending Packages
        self.simulation_time = 0

    def add_drone(self, drone):
        if isinstance(drone, Drone):
            self.drones[drone.droneId] = drone
            print(f"Drone {drone.droneId} added to the fleet.")
        else:
            print("Invalid object. Only Drone objects can be added to the fleet.")

    def add_package_to_pending(self, package):
        if isinstance(package, Package):
            self.pending_packages.append(package)
            print(f"Package {package.packageId} added to pending list.")
        else:
            print("Invalid object. Only Package objects can be added to the pending list.")

    def dispatchJobs(self):
        """Assigns pending packages to idle drones."""
        idle_drones = [drone for drone in self.drones.values() if drone.getStatus() == 'idle']
        assigned_packages = []

        for package in self.pending_packages:
            for drone in idle_drones:
                if drone.assignPackage(package):
                    assigned_packages.append(package)
                    idle_drones.remove(drone) # Remove the drone from the available list
                    break # Move to the next package

        # Remove assigned packages from the pending list
        for package in assigned_packages:
            self.pending_packages.remove(package)

    def simulationTick(self):
        """Simulates one time step, updating drone states."""
        self.simulation_time += 1
        print(f"\n--- Simulation Tick: {self.simulation_time} ---")
        for drone in self.drones.values():
            if drone.getStatus() == 'delivering':
                # Simple simulation: a delivering drone becomes charging after 2 ticks
                if hasattr(drone, 'delivery_time') and self.simulation_time - drone.delivery_time >= 2:
                    drone.setStatus('charging')
                    print(f"Drone {drone.droneId} finished delivery")
```

```

        and is now charging.")
                drone.current_package = None # Package is delivered
            elif not hasattr(drone, 'delivery_time'):
                # Record the start time of delivery for this tick
                    drone.delivery_time = self.simulation_time
                    print(f"Drone {drone.droneId} is delivering package {drone.current_package.packageId}.")
            else:
                print(f"Drone {drone.droneId} is delivering package {drone.current_package.packageId}.")

            elif drone.getStatus() == 'charging':
                # Simple simulation: a charging drone becomes idle after 3 ticks
                    if hasattr(drone, 'charging_time') and self.simulation_time - drone.charging_time >= 3:
                        drone.setStatus('idle')
                        print(f"Drone {drone.droneId} finished charging and is now idle.")
                    del drone.charging_time # Remove charging time attribute
            elif not hasattr(drone, 'charging_time'):
                # Record the start time of charging for this tick
                    drone.charging_time = self.simulation_time
                    print(f"Drone {drone.droneId} is charging.")
            else:
                print(f"Drone {drone.droneId} is charging.")

            elif drone.getStatus() == 'idle':
                print(f"Drone {drone.droneId} is idle.")

    def __repr__(self):
        return f"FleetManager(num_drones={len(self.drones)}, num_pending_packages={len(self.pending_packages)})"

```

## 1. Simulation:

- Implement FleetManager.dispatchJobs() , which assigns packages to idle drones.
- Implement simulationTick() , which loops through all drones and deterministically updates their state (e.g., a 'delivering' drone with a timer of 2 will become 'charging' after 2 ticks).

```
In [15]: # Create a FleetManager
manager = FleetManager()

# Add some drones to the fleet
manager.add_drone(Drone(droneId="D001", maxLoadInKg=10))
manager.add_drone(Drone(droneId="D002", maxLoadInKg=15))
manager.add_drone(Drone(droneId="D003", maxLoadInKg=12))

# Add some packages to the pending list
manager.add_package_to_pending(Package(packageId="P001", weightInKg=8))
manager.add_package_to_pending(Package(packageId="P002", weightInKg=10))
manager.add_package_to_pending(Package(packageId="P003", weightInKg=5))
manager.add_package_to_pending(Package(packageId="P004", weightInKg=18)) # Too heavy for any drone

# Run the simulation for a few ticks
for _ in range(10):
    manager.dispatchJobs()
    manager.simulationTick()
```

Drone D001 added to the fleet.  
Drone D002 added to the fleet.  
Drone D003 added to the fleet.  
Package P001 added to pending list.  
Package P002 added to pending list.  
Package P003 added to pending list.  
Package P004 added to pending list.  
Package P001 assigned to Drone D001  
Package P002 assigned to Drone D002  
Package P003 assigned to Drone D003

--- Simulation Tick: 1 ---  
Drone D001 is delivering package P001.  
Drone D002 is delivering package P002.  
Drone D003 is delivering package P003.

--- Simulation Tick: 2 ---  
Drone D001 is delivering package P001.  
Drone D002 is delivering package P002.  
Drone D003 is delivering package P003.

--- Simulation Tick: 3 ---  
Drone D001 finished delivery and is now charging.  
Drone D002 finished delivery and is now charging.  
Drone D003 finished delivery and is now charging.

--- Simulation Tick: 4 ---  
Drone D001 is charging.  
Drone D002 is charging.  
Drone D003 is charging.

--- Simulation Tick: 5 ---  
Drone D001 is charging.  
Drone D002 is charging.  
Drone D003 is charging.

--- Simulation Tick: 6 ---  
Drone D001 is charging.  
Drone D002 is charging.  
Drone D003 is charging.

--- Simulation Tick: 7 ---  
Drone D001 finished charging and is now idle.  
Drone D002 finished charging and is now idle.  
Drone D003 finished charging and is now idle.  
Package P004 exceeds Drone D001's max load.  
Package P004 exceeds Drone D002's max load.  
Package P004 exceeds Drone D003's max load.

--- Simulation Tick: 8 ---  
Drone D001 is idle.  
Drone D002 is idle.  
Drone D003 is idle.  
Package P004 exceeds Drone D001's max load.  
Package P004 exceeds Drone D002's max load.  
Package P004 exceeds Drone D003's max load.

```
--- Simulation Tick: 9 ---
Drone D001 is idle.
Drone D002 is idle.
Drone D003 is idle.
Package P004 exceeds Drone D001's max load.
Package P004 exceeds Drone D002's max load.
Package P004 exceeds Drone D003's max load.
```

```
--- Simulation Tick: 10 ---
Drone D001 is idle.
Drone D002 is idle.
Drone D003 is idle.
```