Inheritance Exercise

Exercise 1: Vehicle Management System

1. **Base class:** Vehicle
    - Instance variables: brand, model
    - Constructor: initialize both and print "Vehicle constructor called".
2. **Derived class:** Car *(extends Vehicle)*
    - Instance variable: numberOfDoors
    - Constructor: use super() to call the Vehicle constructor and print "Car constructor called".
3. **Method:** displayDetails() in Car that prints all details.
4. **Main method:**
    - Create a Car object with sample data.
    - Call displayDetails().

Exercise 2: University Staff System

1. **Base class:** Staff
    - Instance variables: staffId, name
    - Constructor: initialize variables, print "Staff constructor called".
2. **Derived class:** Professor *(extends Staff)*
    - Instance variable: department
    - Constructor: use super() and print "Professor constructor called".
3. **Method:** displayInfo() to print all details.
4. **Main method:**
    - Create a Professor object with sample data.
    - Display info.

Exercise 3: Flight Reservation System

Base class: Flight

- **Instance variables:**

- flightNumber (int)
- distance (double, in kilometers)
- baseFare (double)

- **Constructor:**
  - Initialize all variables and print **"Flight constructor called"**.
- **Method:**
  - calculateFare() → computes the total fare as distance × baseFare and returns it.

Derived class 1: DomesticFlight ***(extends Flight)***

- **Instance variable:**
  - serviceCharge (double)
- **Constructor:**
  - Call the parent constructor using super() and print **"DomesticFlight constructor called"**.
- **Method Overriding:**
  - Override calculateFare() to include a smaller fixed service charge:
    - Call super.calculateFare() for the base amount.
    - Add serviceCharge.
    - Print **"Domestic flight fare calculated including service charge."**
    - Return the total fare.

Derived class 1: InternationalFlight *(extends Flight)*

- Instance variable:
  - internationalTax (double)
- Constructor:
  - Call the parent constructor using super() and print "InternationalFlight constructor called".
- Method Overriding:
  - Override calculateFare() to include international tax:
    - Call super.calculateFare() to get the base amount.

- Add internationalTax to it.
- Print "International flight fare calculated including international tax."
- Return the total fare.

Main method tasks:

1. Create **one InternationalFlight** and **one DomesticFlight** object using sample data (e.g.,
    - International: flightNumber = 1050, distance = 2500, baseFare = 0.25, internationalTax = 500.
    - Domestic: flightNumber = 215, distance = 800, baseFare = 0.20, serviceCharge = 100.)
2. For both objects:
    - Observe constructor messages showing **hierarchical constructor chaining** (Flight → subclass).
    - Call calculateFare() on each and print the **final fare amount**.
3. Explain how **method overriding** lets each subclass modify the parent's logic differently while reusing the super.calculateFare() computation.

Exercise 4: Vehicle Registration System

1) Base class: Vehicle

- **Variables:** vehicleNumber (String), ownerName (String)
- **Constructor:** initialize both and print **"Vehicle constructor called"**
- **Method (calculation):** computeRegistrationFee()
    - Returns a **base fee** (e.g., 1000.0).
    - Print **"Vehicle: computing base registration fee."**

2) Intermediate class: Car *(extends Vehicle)*

- **Variable:** carType (e.g., "Sedan", "SUV")
- **Constructor:** uses super(...) and prints **"Car constructor called"**
- **Override (calculation):** computeRegistrationFee()

- Start with super.computeRegistrationFee().
- **If** carType is "SUV", add +300; **else** add +150.
- Print **"Car: fee adjusted by car type."**

3) intermediate class: SmartCar *(extends Car)*

- **Variable:** autopilotLevel (int, e.g., 0–3)
- **Constructor:** uses super(...) and prints **"SmartCar constructor called"**
- **Override (calculation):** computeRegistrationFee()
  - Start with super.computeRegistrationFee().
  - Add a **software/ADAS maintenance surcharge**: + (autopilotLevel * 100).
  - Print **"SmartCar: fee adjusted by software/ADAS level."**

4) Derived class: ElectricCar *(extends SmartCar)*

- **Variable:** batteryCapacity (double, kWh)
- **Constructor:** uses super(...) and prints **"ElectricCar constructor called"**
- **Override (calculation):** computeRegistrationFee()
  - Start with super.computeRegistrationFee().
  - Apply **eco adjustment**:
    - Add **road tax per kWh**: + (batteryCapacity * 2)
    - Apply **eco rebate**: - 400
  - Print **"ElectricCar: fee adjusted by battery capacity and eco rebate."**
- **Display method:** showDetails()
  - Show all details (vehicleNumber, ownerName, carType, autopilotLevel, batteryCapacity).
  - Mention the fee policy (type-based + ADAS + EV adjustments).
  - Optionally indicate the **final fee** by calling the calculation method before/after the details.

5) main() tasks

1. **Instantiate** an ElectricCar with sample values, e.g.:
   - vehicleNumber = "DHA-1234"

- ownerName = "Sakib"
- carType = "SUV"
- autopilotLevel = 2
- batteryCapacity = 75.0

2. **Verify constructor order (prints):**
   - "Vehicle constructor called" → "Car constructor called" → "SmartCar constructor called" → "ElectricCar constructor called".
3. **Call** showDetails() on the ElectricCar object.
4. **Call** computeRegistrationFee() and note the **override chain** (prints from each class), and compute the final amount using the layered adjustments.

Exercise 5: University Management System

Base class: Person

- **Instance variables:**
  - name (String)
  - age (int)
- **Constructor:**
  - Initializes both variables and prints **"Person constructor called"**.
- **Methods:**

1. calculateAllowance() → returns base allowance of **1000**.
   - Print **"Base allowance calculated in Person class."**
2. getRole() → returns **"General Person"**.
3. displayInfo() → prints name and age only.

Derived class 1: Student (extends Person)

- **Instance variable:**
  - studentId (String)
- **Constructor:**

- Calls super(name, age) and prints **"Student constructor called"**.

- **Overridden methods:**

1. calculateAllowance() →
    - Calls super.calculateAllowance() and **adds +500** for study materials.
    - Print **"Student allowance includes study materials."**
2. getRole() → returns **"Student"**.
3. displayInfo() →
    - Calls super.displayInfo() to print name and age.
    - Adds studentId and prints **"Displaying Student information."**
4. getDailySchedule() → new method printing a message like **"Student attends classes and library sessions."**

Derived class 2: Teacher (extends Person)

- **Instance variable:**
    - subject (String)
- **Constructor:**
    - Calls super(name, age) and prints **"Teacher constructor called"**.
- **Overridden methods:**

1. calculateAllowance() →
    - Calls super.calculateAllowance() and **adds +800** as teaching bonus.
    - Print **"Teacher allowance includes teaching bonus."**
2. getRole() → returns **"Teacher"**.
3. displayInfo() →
    - Calls super.displayInfo() for base info.
    - Adds subject and prints **"Displaying Teacher information."**
4. getDailySchedule() →
    - Print **"Teacher conducts lectures, grading, and student advising."**

Derived class 3: Staff (extends Person)

- **Instance variable:**
  - department (String)
- **Constructor:**
  - Calls super(name, age) and prints **"Staff constructor called"**.
- **Overridden methods:**

1. calculateAllowance() →
   - Calls super.calculateAllowance() and **adds +300** operational allowance.
   - Print **"Staff allowance includes operational bonus."**
2. getRole() → returns **"Administrative Staff"**.
3. displayInfo() →
   - Calls super.displayInfo() for base info.
   - Adds department and prints **"Displaying Staff information."**
4. getDailySchedule() →
   - Print **"Staff handles office management and administrative support."**

Main method tasks

1. Create **one Student**, **one Teacher**, and **one Staff** object using sample data.
2. Observe constructor chaining sequence:
   "Person constructor called" → "Student/Teacher/Staff constructor called".
3. For each object:
   - Call displayInfo() to view all inherited + subclass data.
   - Call calculateAllowance() to see how each subclass customizes the base logic differently using super.
   - Call getRole() to show overridden identity behavior.
   - Call getDailySchedule() to display their specific daily routines.