Exam 2 Version 1

Coding Portion:

Tool Rental System

You are building a system for renting out tools. Each tool has a base rental cost per day. Some tools have a maintenance fee, others may offer discounts.

- Create one .java file per class/interface/driver
- Format all prices to two decimal places
- You do not need to implement getters/setters/ or use documentation

You Must Implement:

- **RentableLastName.java**
  - An interface which declares a method (calculateRentalPrice) that determines the total rental price based on the number of days the tool is rented.
- **ToolLastName.java**
  - An object that implements RentableLastName.
  - A private String data field named name
  - A private double data field named basePrice
  - A constructor that creates a tool with the specified name and base price
  - A method named getToolInfo that returns a formatted string with the tool name and base price
  - Methods named calculateRentalPrice that accepts either an int days parameter and returns the rental price (basePrice * days) or accepts int days and boolean weekendIncluded as parameters and returns the rental price with a 10% surcharge added if weekendIncluded is true
- **DrillLastName.java**
  - A child of object ToolLastName
  - Overrides the calculateRentalPrice method to add a flat $5.00 maintenance fee to the total rental price.
- **LadderLastName.java**
  - A child of object ToolLastName
  - Overrides the calculateRentalPrice method to apply a 10% discount to the total rental price.
- **ToolRentalDriver.java**
  - Creates two instances of the DrillLastName class with different base prices
  - Creates one instance of the LadderLastName class

- Stores the three tools in an array of ToolLastName
- For each tool in the array, display the tool information and both versions of the rental price:
    - 3-day rental without weekend
    - 3-day rental with weekend (weekendIncluded = true)

Exam 2 Version 2:

Coding Portion:

Appliance Power Usage System

You are building a system that tracks how much power various home appliances use. Each appliance has a power rating (watts per minute) and calculates energy consumption based on usage duration. Some appliances have modified consumption behavior depending on eco mode or appliance type.

- Create one .java file per class/interface/driver
- Format all outputs to two decimal places
- You do not need to implement getters/setters/ or use documentation

You Must Implement:

- **PowerConsumableLastName.java**
  - An interface which declares a method (calculatePowerUsage) that determines power consumption based on how many minutes the appliance is used.
- **ApplianceLastName.java**
  - An object that implements PowerConsumableLastName
  - A private String data field named brand
  - A private double data field named watts (represents power usage per minute)
  - A constructor that creates an appliance with a specified brand and watt usage per minute
  - A method named getApplianceInfo that returns a formatted string with the appliance name and watt usage per minute.
  - Methods named calculatePowerUsage that accepts either an int minutes parameter and returns power usage or accepts int minutes and a boolean ecoMode. If ecoMode is true, total power usage should be reduced by 20%
- **FanLastName.java**
  - A subclass of ApplianceLastName
  - Overrides the power usage calculation for normal mode, power usage is reduced by 10% from the base rate (ecoMode does not apply further reduction).
- **HeaterLastName.java**
  - A subclass of ApplianceLastName
  - Overrides the power usage calculation for normal mode (int minutes), power usage is increased by 20% from the base rate (ecoMode does not increase)

- **ApplianceDriver.java**
    - Creates two instances of FanLastName with different brands and watt ratings
    - Creates one instance of HeaterLastName
    - Stores all three appliances in an array of type ApplianceLastName
    - For each appliance in the array:
        - Print the appliance information
        - Print the power usage for 60 minutes
        - Print the power usage for 60 minutes with eco mode enabled

Exam 2 Version 3

Coding Portion:

Transport Fleet System

You are building a system to model a fleet of transport vehicles. Each vehicle charges a base fare per kilometer. Some vehicles have higher or lower base rates, and fares may increase during rush hour.

- Format all outputs to two decimal places
- Create one .java file per class/interface/driver
- You do not need to implement getters/setters or use documentation

You Must Implement:

- **FareCalculatorLastName.java**
  - An interface that declares a method used to calculate fare based on the distance traveled in kilometers.
- **TransportLastName.java**
  - An object that implements FareCalculatorLastName
  - A private string data field named vehicleID
  - A private string data field named driverName
  - A constructor to initialize the vehicle ID and driver name
  - A method named getTransportInfo that returns a formatted string with the vehicle ID and driver name
  - Methods named calculateFare that either accepts double distance and calculates fare based on a default base rate (distance * 1.25) or accepts double distance and boolean isRushHour and increases the total fare by 25% if isRushHour is true.
- **TaxiLastName.java**
  - A subclass of TransportLastName
  - Overrides the calculateFare method to apply a standard taxi rate (distance * 1.75)
- **PrivateDriverLastName.java**
  - A subclass of TransportLastName
  - Overrides the calculateFare method to apply a luxury rate (distance * 2.5)
- **TransportDriver.java**
  - Creates one instance of TaxiLastName and two instances of PrivateDriverLastName, each with different vehicle IDs and driver names
  - Stores all three in an array of type TransportLastName

- For each vehicle:
    - Print the transport information
    - Print the fare for a 10-kilometer ride
    - Print the fare for a 10-kilometer ride during rush hour (isRushHour = true)