OOP Scenario Set

Scenario 1: Bank Account Management System

A bank is digitizing its customer account system. Design a class-based structure to support both savings and current accounts.

Create a base class called **BankAccount** with the following fields:

- accountNumber (int)

- holderName (string)

- balance (double)

Also include a static field:

- totalAccounts (int)

For each field (including static), write properties.

Add a method:

- ShowDetails() to display all basic fields

Create a class **SavingsAccount** that inherits from **BankAccount** and has the following fields:

- interestRate (double)

- minimumBalance (double)

Write properties for each field and override the ShowDetails() method to include all values.

Add methods:

- CalculateAnnualInterest() → returns balance * interestRate

- IsPenaltyApplicable() → returns true if balance < minimumBalance

Create a class **CurrentAccount** that also inherits from **BankAccount** with fields:

- overdraftLimit (double)

- transactionFee (double)

Add properties for each field and override the ShowDetails() method.

Add overloaded methods:

- CanWithdraw(double amount) → returns true if amount <= balance + overdraftLimit

- CanWithdraw(double amount, double fee) → returns true if amount <= (balance - fee + overdraftLimit)

Write both a default and a parameterized constructor for each class. Create at least two objects (one using default and one using parameterized constructor) per class to demonstrate all the functionality.

Scenario 2: Retail Product Management System

A retail software platform is building a product management module. Design a class structure to manage different types of products.

Create a base class **Product** with the following fields:

- productId (int)

- name (string)

- price (double)

Also include a static field:

- vatRate (double), default value 0.1

For each field (including static), write properties.

Add a method:

- ShowDetails() to display product information

Create a class **DiscountedProduct** that inherits from **Product** and has fields:

- discountPercent (double)

- category (string)

Write properties and override ShowDetails().

Add methods:

- GetFinalPrice() → apply discount and VAT: (price - discount) + VAT

- HasSeasonalOffer() → returns true if category is "Festival" or "Winter"

Create a class **PremiumProduct** that also inherits from **Product** and has fields:

- warrantyYears (int)

- countryOfOrigin (string)

Add properties and override ShowDetails().

Add methods:

- IsEligibleForVIPReturn() → returns true if warrantyYears >= 2

Add overloaded methods:

- ShowWarrantyInfo() → returns a basic message like "Warranty: X years"

- ShowWarrantyInfo(string note) → returns a message with the note included, e.g., "Warranty: X years (note)"

Write both a default and a parameterized constructor for each class. Create at least two objects (one using default and one using parameterized constructor) per class to demonstrate all the functionality.

Scenario 3: Shape Drawing System

A shape modeling tool needs to support drawing and calculations for different shapes.

Create a base class **Shape** with the following fields:

- shapeName (string)

- shapeType (string)

Also include a static field:

- totalShapes (int)

For each field (including static), write properties.

Add a method:

- ShowDetails()

Create a class **Rectangle** that inherits from **Shape** and has the following field:

- length (double)

- width (double)

Add properties and override ShowDetails().

Add methods:

- IsSquare() → true if length == width

Add overloaded methods:

- GetPerimeter() → returns the perimeter calculated as 2 * (length + width)

- GetPerimeter(string unit) → returns a formatted message including the perimeter and the unit, e.g., "Perimeter: 20 cm"

Create a class **Circle** that also inherits from **Shape** and has the following field:

- radius (double)

- color (string)

Add properties and override ShowDetails().

Add methods:

- GetDiameter() → returns radius * 2

- IsLargeCircle() → true if radius > 10

Write both a default and a parameterized constructor for each class. Create at least two objects (one using default and one using parameterized constructor) per class to demonstrate all the functionality.

Scenario 4: Ticket Booking System

A booking system is being built to manage tickets for trains and flights.

Create a base class **Ticket** with the following fields:

- ticketId (int)

- passengerName (string)

Static field:

- totalBookings (int)

For each field (including static), write properties.

Add a method:

- ShowDetails()

Create class **TrainTicket** that inherits from **Ticket** and has the following field:

- coachType (string)

- fare (double)

Add properties and override ShowDetails().

Add methods:

- IsACCoach() → true if coachType == "AC"

- GetTaxAmount() → returns fare * 0.10

Create class **FlightTicket** that also inherits from **Ticket** and has the following field:

- seatClass (string)

- luggageWeight (double)

Add properties and override ShowDetails().

Add methods:

- IsExcessLuggage() → true if luggageWeight > 20

Add overloaded methods:

- IsPrioritySeat() → returns true if seatClass is "Business"

- IsPrioritySeat(bool isFrequentFlyer) → returns true only if isFrequentFlyer is true AND seatClass is "Business"

Write both a default and a parameterized constructor for each class. Create at least two objects (one using default and one using parameterized constructor) per class to demonstrate all the functionality.

Scenario 5: Appliance Energy Monitoring System

A home energy app tracks different electrical appliances and their energy usage.

Create a base class **Appliance** with the following fields:

- applianceName (string)

- brand (string)

Static field:

- totalEnergyUsed (double)

For each field (including static), write properties.

Add Method:

- ShowDetails().

Create **Fan** subclass that inherits from **Appliance** and has the following field:

- bladeSize (int)

- speedLevels (int)

Add properties and override ShowDetails().

Add methods:

- MaxCoolingEfficiency() → return bladeSize * speedLevels

- NoiseCategory() → returns: - "Low" if bladeSize < 10 - "Medium" if 10 ≤ bladeSize ≤ 15 - "High" if bladeSize > 15

Create **AC** subclass that also inherits from **Appliance** and has the following field:

- btu (int)

- starRating (int)

Add properties and override ShowDetails().

Add overloaded methods:

- MonthlyElectricityCost(int hours) → returns btu * hours * 0.05

- MonthlyElectricityCost(int hours, double unitCost) → adds result to totalEnergyUsed

Add method:

- IsEcoFriendly() → true if starRating >= 4

Write both a default and a parameterized constructor for each class. Create at least two objects (one using default and one using parameterized constructor) per class to demonstrate all the functionality.