CASE STUDY 2: Online Food Ordering & Cart Billing System

Case Study Description

A startup called **FoodGo** wants a **food ordering and billing system**. Customers can add food items to their cart, apply discounts, and generate bills automatically.

The system must include:

- **Classes:** `FoodItem`, `Cart`, `PremiumFoodItem` (inheritance)
- **Methods & validation** with error handling
- **Custom errors** for invalid input
- **Timer logic:** `setTimeout` to simulate order preparation
- **Closures** for discount calculation
- **Spread operator** for adding multiple items
- **Arrow functions** for internal calculations
- **Type conversion** for price validation

Step-by-Step Implementation (No Code)

1. Create Classes

a) FoodItem Class

- Represents a single food item.
- **Properties needed:**
  - `name`: the name of the food item (string)
  - `price`: cost of the food item (number)
  - `category`: type of food, e.g., Fast Food, Regular (string)
- **Responsibilities:**
  - Validate that `name` and `price` are provided

    - Ensure `price` is a valid positive number
    - Assign default category if not provided
- **Why:** This ensures each food item has correct and complete information.

b) PremiumFoodItem Class (Inheritance)

- Represents premium items with extra features (like toppings).
- **Extra Property:** `extraFee` (additional cost for premium items)
- **Responsibilities:**
    - Inherit properties from `FoodItem`
    - Add the `extraFee` to the total price
- **Why:** Demonstrates inheritance and method overriding.

c) Cart Class

- Represents the customer's shopping cart.
- **Properties:** `items` → an array to store all items added.
- **Responsibilities:**
    - Add multiple items to the cart (using rest operator)
    - Calculate the total price of all items in the cart
    - Check if each item is valid before adding
- **Why:** Teaches object composition and working with arrays.

2. Use Constructors, Methods, Static Properties, and Error Handling

Constructors

- Initialize the object with necessary properties.
- Each class (`FoodItem`, `PremiumFoodItem`, `Cart`) should have a constructor.

Methods

- `getTotalPrice` for premium items

- `addItems` and `calculateTotal` for the cart
- **Why:** Methods allow objects to perform actions and calculations.

Static Properties

- Optional property to hold shared information (e.g., company name)
- Accessible without creating an instance.

Error Handling

- Use `try/catch` to handle invalid data.
- Create **custom errors** (like `ValidationError`) to notify users when something is wrong.

3. Billing Logic

a) Calculate Total Price

- Sum up the prices of all items in the cart.
- Include `extraFee` for premium items.

b) Apply Discounts Using Closure

- Create a closure function that stores the discount percentage.
- Apply it to the total price to calculate the discounted price.
- **Why:** Demonstrates how closures "remember" values and can be reused.

c) Simulate Billing Using Timer

- Use `setTimeout` to simulate food preparation or billing delay.
- Print a detailed bill including:
    - Names of all items in the cart
    - Total price before discount
    - Total price after discount

d) Detailed Bill Summary

- Clearly show:
    - Item names

    - Original total price
    - Discounted total price
- Helps customers understand what they are paying for.

4. Execution Flow for Beginners

1. **Create food items** (some regular, some premium).
2. **Create a cart** object.
3. **Add items to the cart** (validate each item).
4. **Calculate the total price** of all items.
5. **Apply discount** using the closure function.
6. **Simulate preparation time** using a timer.
7. **Print the final bill** with item names, original price, and discounted price.