Inventory Reordering System

1. Introduction

The Inventory Reordering System is designed to ensure that warehouse inventory levels meet forecasted demand while minimizing reordering costs. The system calculates the number of units to reorder for each item based on its current stock, forecasted demand, reorder cost per unit, and batch size.

2. Algorithm Overview

The algorithm works as follows:

- **Input:** A list of inventory items with attributes such as ***ItemId***, ***CurrentStock***, ***ForecastedDemand***, ***ReorderCostPerUnit***, and ***BatchSize***.
- **Output:** A reordering plan specifying the ***ItemId*** and the number of units to reorder for each item.

The algorithm calculates the required units to meet the forecasted demand and ensures that the reorder quantity is a multiple of the batch size to minimize costs.

3. Code Explanation

InventoryItem Class

- **Properties**
  - **ItemId**: Unique identifier for the item.
  - **CurrentStock**: Current inventory level of the item.
  - **ForecastedDemand**: Forecasted demand for the next period.
  - **ReorderCostPerUnit**: Cost incurred for ordering a single unit of the item.
  - **BatchSize**: The number of units that must be ordered in a single batch.

- **Methods:**
    - **CalculateUnitsToOrder()**:
        - Calculates the number of units required to meet the forecasted demand.
        - If the current stock is sufficient, no units are ordered.
        - If additional units are needed, the method calculates the number of batches required and returns the total units to order.

Program Class

- **Methods:**
    - **GenerateReorderingPlan(List<InventoryItem> inventoryItems)**:
        - Takes a list of `InventoryItem` objects as input.
        - Iterates through each item and calculates the units to order using the `CalculateUnitsToOrder()` method.
        - Returns a list of tuples containing the `ItemId` and the number of units to order.
    - **Main(string[] args)**:
        - Initializes a list of `InventoryItem` objects with sample data.
        - Calls the `GenerateReorderingPlan()` method to generate the reordering plan.
        - Outputs the reordering plan to the console.

4. **Sample Run**

Input Data:

```
List<InventoryItem> inventoryItems = new List<InventoryItem>
{
    new InventoryItem(itemId: 1, currentStock: 50, forecastedDemand: 100, reorderCostPerUnit: 10, batchSize: 20),
```

```
    new InventoryItem(itemId: 2, currentStock: 30, forecastedDemand: 60, reorderCostPerUnit: 5, batchSize: 15),
    new InventoryItem(itemId: 3, currentStock: 80, forecastedDemand: 70, reorderCostPerUnit: 8, batchSize: 10),
    new InventoryItem(itemId: 4, currentStock: 10, forecastedDemand: 50, reorderCostPerUnit: 12, batchSize: 25)
};
```

Output:

```
Reordering Plan:
Item ID: 1, Units to Order: 60
Item ID: 2, Units to Order: 30
Item ID: 4, Units to Order: 50
```

Explanation:

- **Item 1:**
  - Current Stock: 50
  - Forecasted Demand: 100
  - Required Units: 50 (100 - 50)
  - Batch Size: 20
  - Batches Needed: 3 (since 50 / 20 = 2.5, rounded up to 3)
  - Units to Order: 60 (3 * 20)
- **Item 2:**
  - Current Stock: 30
  - Forecasted Demand: 60
  - Required Units: 30 (60 - 30)
  - Batch Size: 15
  - Batches Needed: 2 (30 / 15 = 2)
  - Units to Order: 30 (2 * 15)
- **Item 3:**
  - Current Stock: 80
  - Forecasted Demand: 70
  - Required Units: 0 (since current stock is sufficient)

    - Units to Order: 0
- **Item 4:**
    - Current Stock: 10
    - Forecasted Demand: 50
    - Required Units: 40 (50 - 10)
    - Batch Size: 25
    - Batches Needed: 2 (40 / 25 = 1.6, rounded up to 2)
    - Units to Order: 50 (2 * 25)

5. **Flowchart**

   Below is a simplified flowchart representing the algorithm:

   Start
   |
   Initialize Inventory Items
   |
   For each item in inventory:
   |
   Calculate Required Units = Forecasted Demand - Current Stock
   |
   If Required Units <= 0:
   |
   No reorder needed
   |
   Else:
   |
   Calculate Batches Needed = Ceiling(Required Units / Batch Size)
   |
   Calculate Units to Order = Batches Needed * Batch Size
   |
   Add (ItemId, Units to Order) to Reordering Plan
   |
   End For
   |
   Output Reordering Plan
   |
   End

6. **How to Run the Code**

- **Prerequisites:**
    - Ensure you have the .NET SDK installed on your machine.
    - A code editor like Visual Studio or Visual Studio Code
- **Steps:**
    - The provided code into a new C# Console Application project.
    - Build (>dotnet build) and run (>dotnet run) the project.
    - The console will display the reordering plan based on the sample data.

7. **Conclusion**

The Inventory Reordering System effectively calculates the optimal reorder quantities for each item, ensuring that stock levels meet forecasted demand while minimizing costs. The algorithm is efficient, modular, and easy to understand, making it suitable for real-world warehouse inventory management.