### **Inventory Management System**

**Scenario:**You are tasked with designing an **Inventory Management System** for a warehouse. The system should handle multiple item types, allow for saving and loading inventory data from files, validate user inputs, and be easily extensible for new features or item types. This problem assesses your understanding of OOP, SOLID principles, exception handling, and file I/O.

### **Requirements**

#### **Part 1: Object-Oriented Design**

1. **Base Class:**
   * Create a base class Item with the following properties:
     + Name (string): The name of the item (e.g., "Milk").
     + Quantity (integer): The number of items in stock.
     + Price (float): The price per item in USD.
   * Add a method GetDescription() in Item that returns a string in the format:  
     "Name: {Name}, Quantity: {Quantity}, Price: ${Price}".
2. **Derived Classes:**
   * Create two derived classes:
     + PerishableItem with an additional property ExpiryDate (DateTime).
       - Override the GetDescription() method to include the expiry date:  
         "Name: {Name}, Quantity: {Quantity}, Price: ${Price}, Expiry Date: {ExpiryDate}".
     + NonPerishableItem with no additional properties, but override GetDescription() for consistency.
3. **Extensibility:**Ensure that new item types (e.g., BulkItem) can be added without modifying existing classes by designing the inheritance hierarchy properly.

#### **Part 2: SOLID Principles**

1. **Inventory Repository:**
   * Define an interface IInventoryRepository with the following methods:
     + void SaveToFile(string filePath, List<Item> items): Saves the inventory data to the specified file.
     + List<Item> LoadFromFile(string filePath): Loads inventory data from the specified file and returns a list of Item objects.
2. **Implementation of Repository:**
   * Create a class FileInventoryRepository that implements IInventoryRepository:
     + Use plain text (TXT) as the file format for saving and loading inventory.
     + Ensure that each Item is serialized/deserialized into the appropriate subclass (PerishableItem, NonPerishableItem).
   * Example TXT file:  
     PerishableItem|Milk|10|2.5|2024-01-15
     + NonPerishableItem|Rice|50|1.2
       - The format uses | as a delimiter, with the first field indicating the item type.
3. **Inventory Management:**
   * Create a class InventoryManager with methods:
     + void AddItem(Item item): Adds a new item to the inventory.
     + void UpdateItem(string itemName, int newQuantity): Updates the quantity of an existing item.
     + void RemoveItem(string itemName): Removes an item by name.
     + Item GetItem(string itemName): Retrieves an item by name.
     + List<Item> GetAllItems(): Returns a list of all items in the inventory.
   * Ensure the **Single Responsibility Principle** (SRP) is followed by keeping file operations in FileInventoryRepository and inventory logic in InventoryManager.
4. **Open/Closed Principle (OCP):**
   * Implement new item types (e.g., BulkItem with a Discount property) without modifying existing classes. For example:
     + BulkItem should override GetDescription() to include the discount:  
       "Name: {Name}, Quantity: {Quantity}, Price: ${Price}, Discount: {Discount}%".

#### **Part 3: Exception Handling**

1. **Validation Logic:**
   * Ensure that:
     + Quantity cannot be negative.
     + Price cannot be negative.
     + ExpiryDate (for PerishableItem) must be a future date.
     + Throw an appropriate exception (e.g., ArgumentException) when validation fails.
2. **File Operations:**
   * Add error handling for file-related issues, such as:
     + FileNotFoundException when attempting to load a non-existent file.
     + IOException for unexpected file I/O errors.
   * Log detailed error messages to the console or a log file.
3. **Parsing Errors:**
   * Handle invalid or malformed text data with meaningful error messages.

#### **Part 4: Input/Output**

* + Create a console-based program that allows users to perform the following operations:
    - Load inventory from a TXT file.
    - Add new items or update existing ones via user input.
    - Remove items by name.
    - View all items in the inventory.
    - Save the updated inventory back to the TXT file.