**C# Bootcamp Week 2: Design Patterns Assignment**

**E-Commerce Order Management System**

**Assignment Overview**

You will build a comprehensive E-Commerce Order Management System that demonstrates mastery of key design patterns and SOLID principles. This project simulates a real-world scenario where you'll implement order processing, inventory management, and payment handling using industry-standard design patterns.

**Learning Objectives**

By completing this assignment, you will:

* Implement the Factory, Singleton, Strategy, and Repository patterns
* Apply SOLID principles in practical scenarios
* Design clean, maintainable, and extensible code architecture
* Understand separation of concerns in enterprise applications
* Practice proper use of interfaces and abstractions

**System Requirements**

**Core Functionality**

Your system must support:

1. Product Management: Create, read, update products with different categories
2. Order Processing: Handle customer orders with various payment methods
3. Inventory Tracking: Manage stock levels and availability
4. Payment Processing: Support multiple payment strategies
5. Logging & Configuration: System-wide logging and configuration management

**Technical Requirements**

* Project Type: Console Application with clear menu system
* Architecture: Clean separation of concerns across layers  
  Design Pattern Implementation Requirements

**1. Factory Pattern (Creational)**  
Requirement: Implement a Product Factory to create different product types.  
Implementation Details

public enum ProductCategory

{

Electronics,

Clothing,

Books,

HomeGarden

}

Expected Behavior:

* Factory creates appropriate product types (Electronics, Clothing, Books, HomeGarden)
* Each product type has unique behavior in GetProductDetails()
* Factory handles invalid category gracefully
* Products have category-specific properties (e.g., Electronics have warranty, Books have ISBN)

**2. Singleton Pattern (Creational)**  
Requirement: Create a thread-safe Configuration Manager and Logger.  
Implementation Details:

public interface IConfigurationManager

{

string GetSetting(string key);

void SetSetting(string key, string value);

}

public interface ILogger

{

void LogInfo(string message);

void LogError(string message);

void LogWarning(string message);

}

Expected Behavior:

* Thread-safe singleton implementation
* Configuration manager stores app settings (database connection, API keys, etc.)
* Logger writes to console with timestamps and log levels
* Both singletons properly handle concurrent access

**3. Strategy Pattern (Behavioral)**  
Requirement: Implement multiple payment processing strategies.  
Required Strategies:

* CreditCardPayment: Validates card number, expiry, CVV
* PayPalPayment: Uses email and password validation
* BankTransferPayment: Requires routing and account numbers
* CryptoPayment: Uses wallet address validation  
  Expected Behavior:
* Each strategy has unique validation logic
* Strategies can be swapped at runtime
* Payment context manages strategy selection
* Each payment type has different success/failure scenarios

**4. Repository Pattern (Layered Abstraction)**  
Requirement: Implement repository pattern for data access layer.  
Implementation Details:

public interface IRepository

{

GetByIdAsync(string id);

GetAllAsync();

AddAsync(T entity);

UpdateAsync(T entity);

DeleteAsync(string id);

}

public interface IProductRepository : IRepository

{

Task> GetByCategory (ProductCategory category);

GetLowStockProducts (int threshold);

}

public interface IOrderRepository : IRepository

{

GetOrdersByCustomer(string customerId);

GetOrdersByDateRange(DateTime start, DateTime end);

}

**Expected Behavior:**

* In-memory implementation (no actual database required)
* Generic repository pattern with specific implementations
* Repository abstracts data access from business logic
* SOLID Principles Application

**Project Structure**

ECommerceOrderSystem/

├── Models/

├── Factories/

├── Singletons/

├── Strategies/

├── Repositories/

├── Services/

└── Program.cs

**Sample Application Flow**  
Your console application should provide a menu-driven interface:  
=== E-Commerce Order Management System ===

1. View Products
2. Add New Product
3. Create Order
4. Process Payment
5. View Orders
6. Check Inventory
7. System Logs
8. Configuration
9. Exit  
   Please select an option:

**Deliverables**

Code Submission  
• Complete C# solution with all required patterns  
• Proper error handling and validation  
• Console application with working menu system

**Documentation**

Submit a Design Document (2-3 pages) that includes:

1. Architecture Overview: High-level system design diagram
2. Pattern Implementation: Explanation of how each pattern is implemented
3. SOLID Principles: Examples of how you applied each principle
4. Challenges & Solutions: Problems encountered and how you solved them
5. Future Enhancements: How the system could be extended