**Documentation for Library Management System Code**

**Team members**

**1-mohammed Shaker Salama 21-02042**

**2-Adel fawzy elezaway 21-00261**

**3-Mohamed Ebrahim Elkhalifa 21-01163**

**4- Ahmed Amgad Shasha 21-00404**

**2100261**

**Table of Contents**

1. Introduction
2. Class Overview
   * DatabaseConnection
   * Logger
   * Book and Its Variants
   * BookFactory and FactoryProducer
   * ExternalBook and ExternalBookAdapter
   * LibraryUser and Its Variants
   * BookService and Its Proxy
   * Book Commands
3. Design Patterns
   * Singleton Pattern
   * Prototype Pattern
   * Factory Pattern
   * Adapter Pattern
   * Builder Pattern
   * Proxy Pattern
   * Command Pattern
4. Code Flow
5. Examples
6. Additional Considerations

**1. Introduction**

This document provides a detailed explanation of the library management system code. It highlights the purpose, functionality, and relationships of the classes. Additionally, it elaborates on the design patterns used, their implementation, and their roles in the system.

**2. Class Overview**

**2.1 DatabaseConnection**

* **Purpose:** Implements a Singleton pattern to manage the database connection for the application.
* **Attributes:**
  + instance (static): The single instance of the class.
  + connection: The Connection object for the database.
* **Methods:**
  + getInstance(): Ensures only one instance of the class exists.
  + getConnection(): Provides access to the database connection.

**2.2 Logger**

* **Purpose:** Implements a thread-safe Singleton pattern for logging messages.
* **Attributes:**
  + logger (static): The single instance of the Logger.
* **Methods:**
  + getInstance(): Ensures only one instance of Logger exists.
  + log(String message): Logs messages to the console.

**2.3 Book and Its Variants**

* **Purpose:** Implements the Prototype pattern to create and manage books.
* **Attributes:**
  + title, author, category: Common attributes for all books.
* **Methods:**
  + Getter and Setter methods for attributes.
  + clone(): Creates a copy of the book.
* **Variants:**
  + SoftwareEngineeringBook, ManagementBook, ArtificialIntelligenceBook: Specialized book categories.

**2.4 BookFactory and FactoryProducer**

* **Purpose:** Implements the Factory pattern to create books based on categories.
* **Components:**
  + BookFactory: Interface for creating books.
  + SoftwareEngineeringBookFactory, ManagementBookFactory, ArtificialIntelligenceBookFactory: Concrete factories for each book type.
  + BookFactoryProducer: Factory of factories that returns the appropriate factory based on the category.

**2.5 ExternalBook and ExternalBookAdapter**

* **Purpose:** Implements the Adapter pattern to integrate external books into the system.
* **Components:**
  + ExternalBook: Represents books from an external source.
  + ExternalBookAdapter: Adapts ExternalBook to the system's IBook interface.

**2.6 LibraryUser and Its Variants**

* **Purpose:** Represents users in the library system with an abstract class and specialized subclasses.
* **Components:**
  + LibraryUser: Abstract class with a common name attribute.
  + Admin, RegularUser: Specialized classes for different user roles.

**2.7 BookService and Its Proxy**

* **Purpose:** Implements the Proxy pattern for managing book operations.
* **Components:**
  + RealBookService: Handles direct operations with the database.
  + BookServiceProxy: Logs operations and delegates to RealBookService.

**2.8 Book Commands**

* **Purpose:** Implements the Command pattern for borrowing and returning books.
* **Components:**
  + BorrowBookCommand: Handles the logic for borrowing books.
  + ReturnBookCommand: Handles the logic for returning books.

**3. Design Patterns**

**3.1 Singleton Pattern**

* **Implementation:** Used in DatabaseConnection and Logger to ensure a single instance.
* **Purpose:** Provides centralized control and prevents multiple instances of critical resources.

**3.2 Prototype Pattern**

* **Implementation:** Used in the Book class to clone books.
* **Purpose:** Simplifies the creation of book objects by cloning existing instances.

**3.3 Factory Pattern**

* **Implementation:**
  + BookFactory: Abstract factory interface.
  + Concrete factories for each book category.
  + BookFactoryProducer: Returns the appropriate factory.
* **Purpose:** Decouples the creation logic from the client code.

**3.4 Adapter Pattern**

* **Implementation:**
  + ExternalBookAdapter: Converts ExternalBook to conform to the IBook interface.
* **Purpose:** Allows integration of external books into the library system.

**3.5 Builder Pattern**

* **Implementation:**
  + BookBuilder interface and ConcreteBookBuilder class.
* **Purpose:** Simplifies the construction of complex book objects.

**3.6 Proxy Pattern**

* **Implementation:**
  + BookServiceProxy: Logs operations and delegates to RealBookService.
* **Purpose:** Adds logging and control without modifying the underlying service.

**3.7 Command Pattern**

* **Implementation:**
  + BorrowBookCommand, ReturnBookCommand: Encapsulate book operations as commands.
* **Purpose:** Decouples request logic from execution.

**4. Code Flow**

1. The application initializes the database connection using DatabaseConnection.
2. BookServiceProxy manages book operations, logging actions and delegating to RealBookService.
3. Books are created using the Factory and Builder patterns.
4. External books are integrated via the Adapter pattern.
5. Commands handle user actions like borrowing and returning books.
6. Logger logs significant actions throughout the application.

**5. Examples**

**Singleton Example**

DatabaseConnection dbConnection = DatabaseConnection.getInstance();

Connection conn = dbConnection.getConnection();

**Factory Example**

BookFactory factory = BookFactoryProducer.getFactory("Software Engineering");

Book book = factory.createBook();

**Adapter Example**

ExternalBook externalBook = new ExternalBook("Title", "Author", "Category");

IBook adaptedBook = new ExternalBookAdapter(externalBook);

System.out.println(adaptedBook.getTitle());

**Command Example**

BookCommand borrowCommand = new BorrowBookCommand(bookService, "Book Title", "User Name");

boolean success = borrowCommand.execute();

**6. Additional Considerations**

* **Optimizations:**
  + Lazy initialization in Singleton classes.
  + Efficient database operations with prepared statements.
* **Future Improvements:**
  + Implement caching for frequently accessed books.
  + Add role-based access control for Admin and Regular Users.
  + Refactor BookService to use a Dependency Injection framework.