Library Management System Documentation

# Table of Contents

# 1. Class Overview

## DatabaseConnection

- **Purpose**: Manages a single connection to the database using the Singleton Pattern.
- **Attributes and Methods**:
  - `connection`: Stores the database connection object.
  - `getInstance()`: Returns the single instance of the DatabaseConnection class.
  - `getConnection()`: Provides access to the database connection.
- **Design Patterns**: Singleton

## Logger

- **Purpose**: Provides a thread-safe logging mechanism for the system.
- **Attributes and Methods**:
  - `getInstance()`: Returns the single instance of the Logger class.

  - `log(message)`: Logs a given message.

- **Design Patterns**: Singleton

... (truncated for example purposes)

# 2. Design Patterns

## Singleton Pattern

Implementation: Used in `DatabaseConnection` and `Logger` classes to ensure a single instance exists.

Components:

- Private constructor prevents direct instantiation.
- Static `getInstance()` ensures only one instance is created.

Use Case: Centralized database access and consistent logging.

## Factory Pattern

Implementation: Used to create specific types of books (`SoftwareEngineeringBook`, `ManagementBook`, etc.).

Components:

- Abstract `BookFactory` interface defines `createBook()`.
- Concrete factories implement `createBook()` to return specific book types.

Use Case: Simplifies the creation of book objects based on category.

... (truncated for example purposes)

# 3. Code Flow

1. The `DatabaseConnection` class establishes a database connection.
2. The `BookServiceProxy` manages requests to add, remove, or retrieve books.
3. Design patterns like Factory and Command simplify object creation and book operations.

# 4. Examples

### DatabaseConnection Class Example:
```java
DatabaseConnection db = DatabaseConnection.getInstance();
Connection conn = db.getConnection();
```

### Factory Pattern Example:
```java
BookFactory factory = BookFactoryProducer.getFactory("Software Engineering");
Book book = factory.createBook();
```

...

# 5. Additional Considerations

## Optimizations

- Singleton ensures centralized resource management.
- Factory decouples object creation logic.

## Future Improvements

- Implement Observer Pattern for real-time notifications.
- Add more specific categories in Factory Pattern.