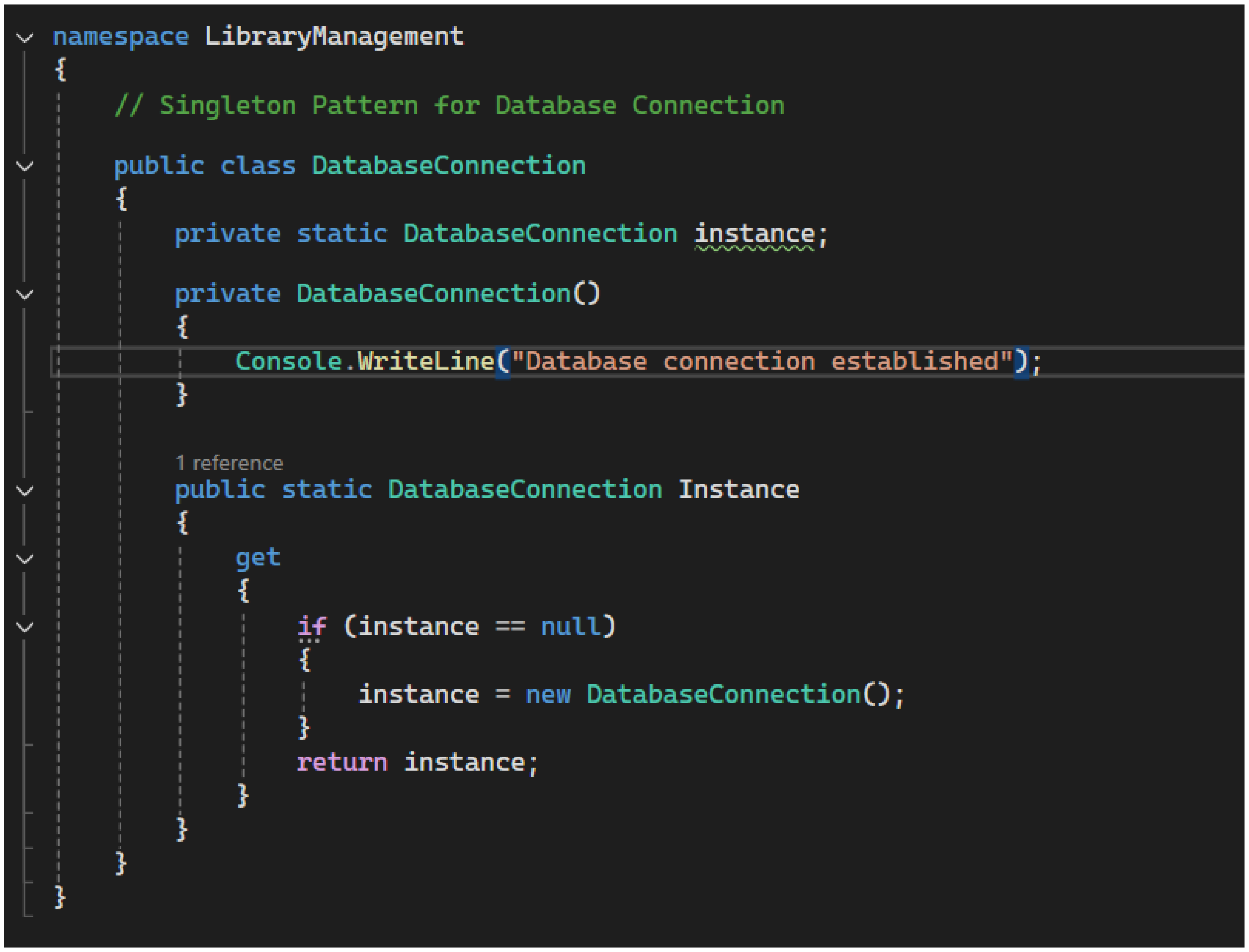
Lab 5 – Design Patterns

I. Applying design patterns:

1. Singleton Pattern Application:

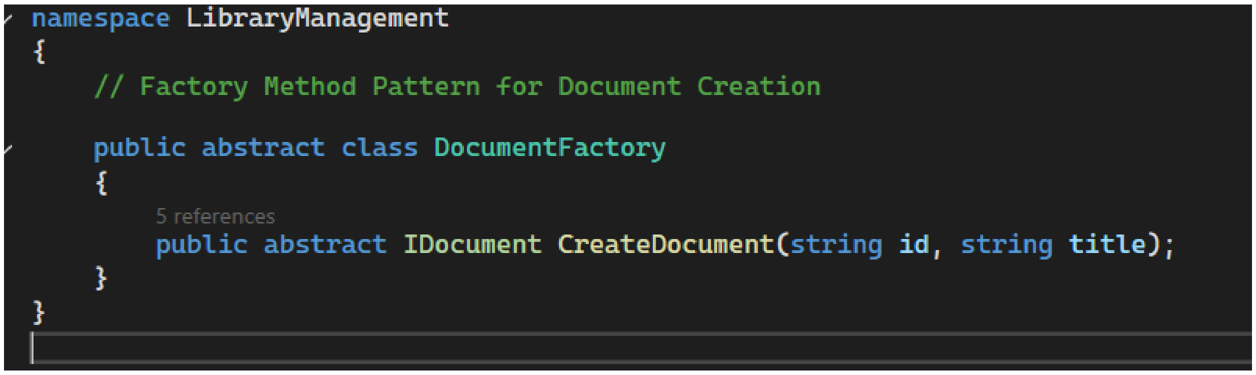


```csharp
namespace LibraryManagement
{
    // Singleton Pattern for Database Connection

    public class DatabaseConnection
    {
        private static DatabaseConnection instance;

        private DatabaseConnection()
        {
            Console.WriteLine("Database connection established");
        }

        1 reference
        public static DatabaseConnection Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new DatabaseConnection();
                }
                return instance;
            }
        }
    }
}
```

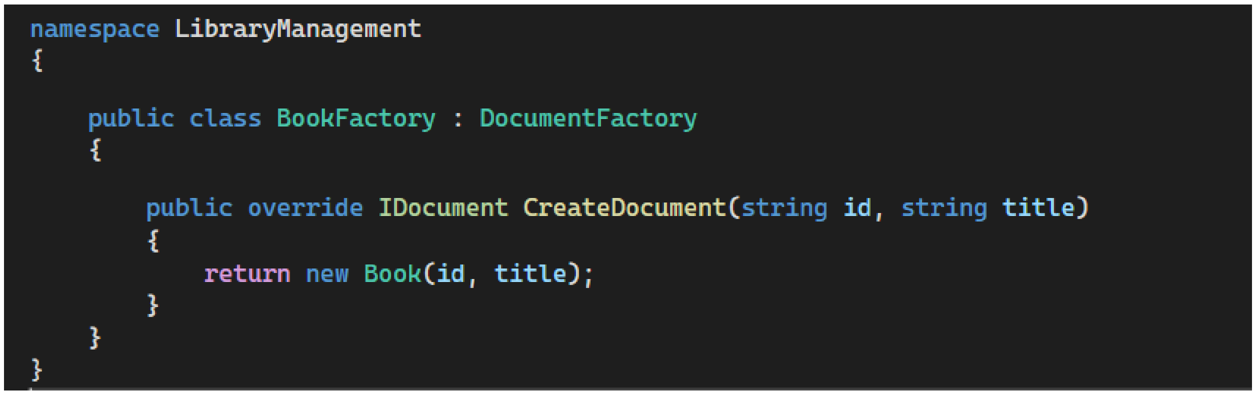
- Implemented in DatabaseConnection class
- Uses private constructor and static Instance property
- Ensures only one database connection instance exists
- Provides thread-safe access through lazy initialization
- Global access point through DatabaseConnection.Instance

2. Factory Method Pattern Application:

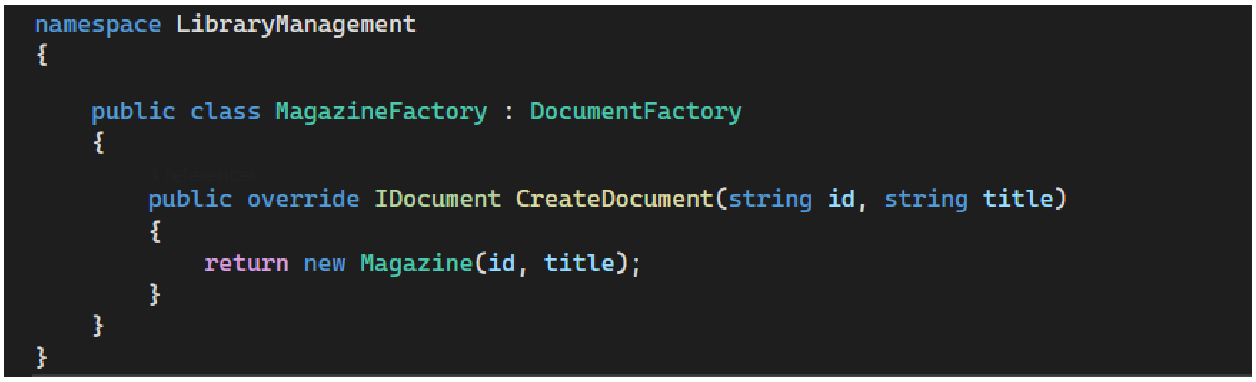
- Implemented through abstract DocumentFactory class and concrete factories



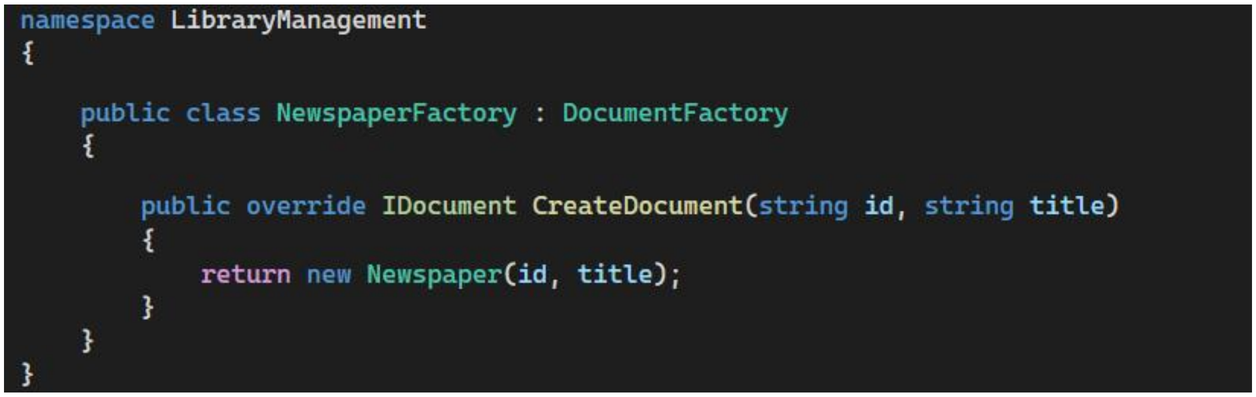
```csharp
namespace LibraryManagement
{
    // Factory Method Pattern for Document Creation

    public abstract class DocumentFactory
    {
        5 references
        public abstract IDocument CreateDocument(string id, string title);
    }
}
```



```csharp
namespace LibraryManagement
{

    public class BookFactory : DocumentFactory
    {

        public override IDocument CreateDocument(string id, string title)
        {
            return new Book(id, title);
        }
    }
}
```



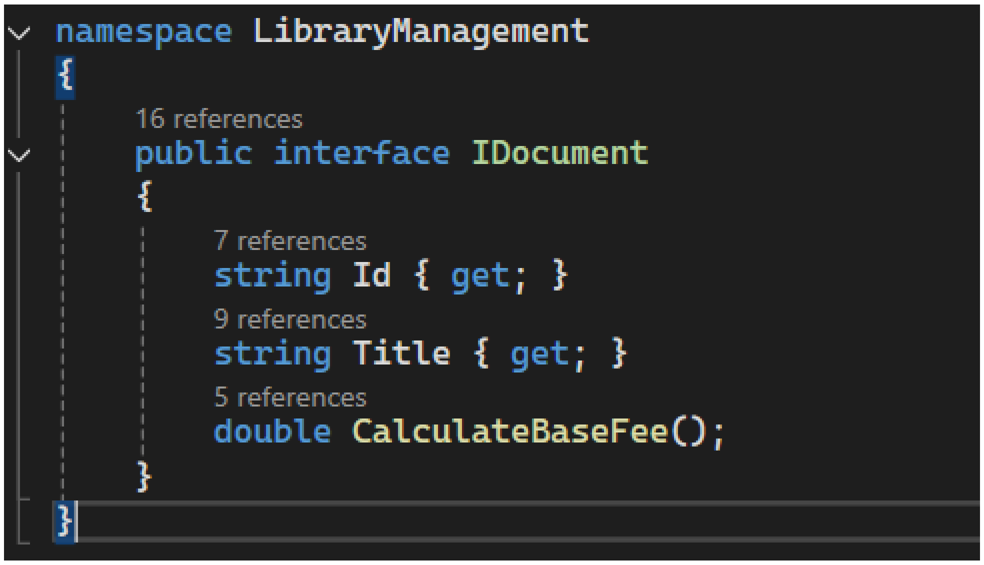
```csharp
namespace LibraryManagement
{

    public class MagazineFactory : DocumentFactory
    {

        public override IDocument CreateDocument(string id, string title)
        {
            return new Magazine(id, title);
        }
    }
}
```



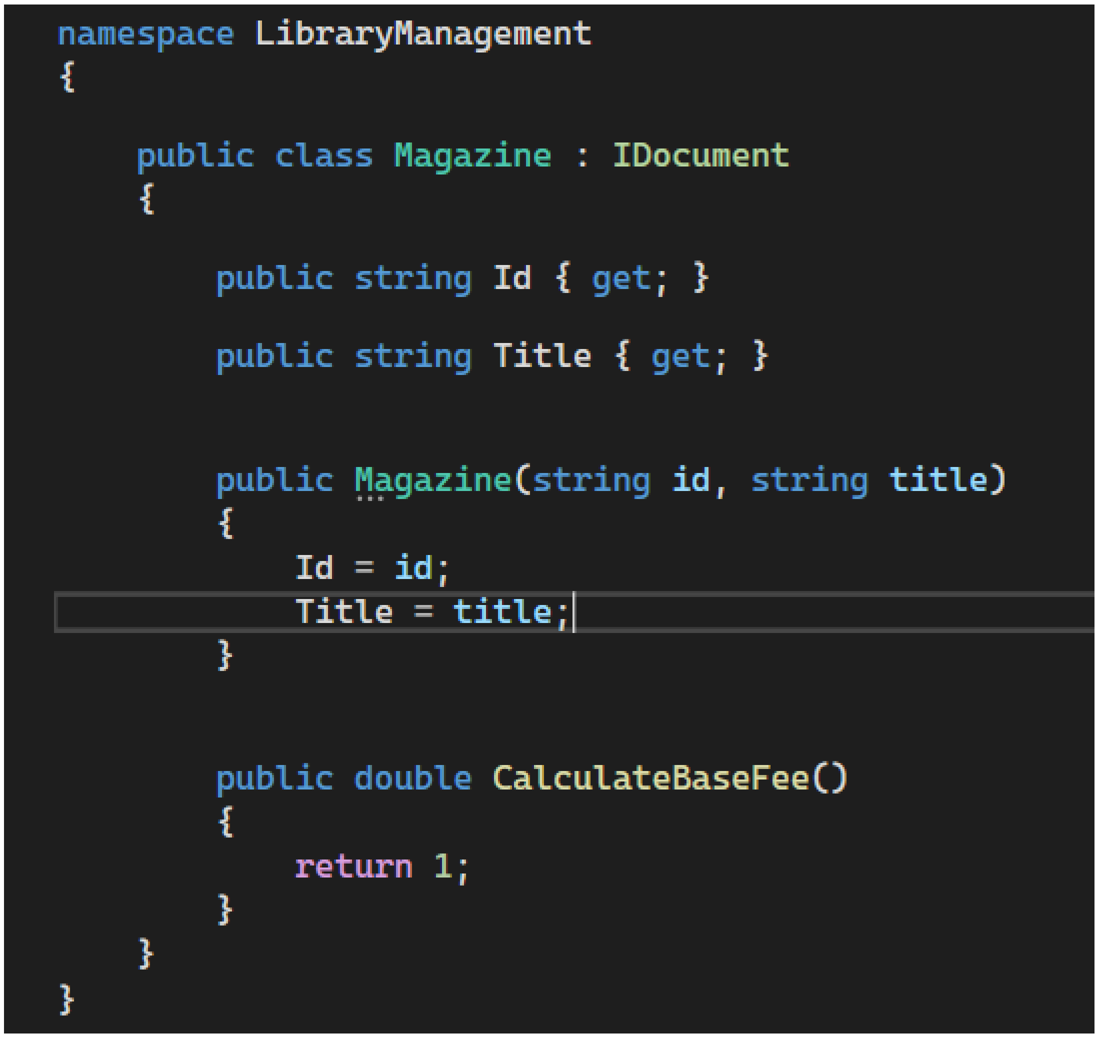
```csharp
namespace LibraryManagement
{

    public class NewspaperFactory : DocumentFactory
    {

        public override IDocument CreateDocument(string id, string title)
        {
            return new Newspaper(id, title);
        }
    }
}
```

- BookFactory, MagazineFactory, and NewspaperFactory subclasses

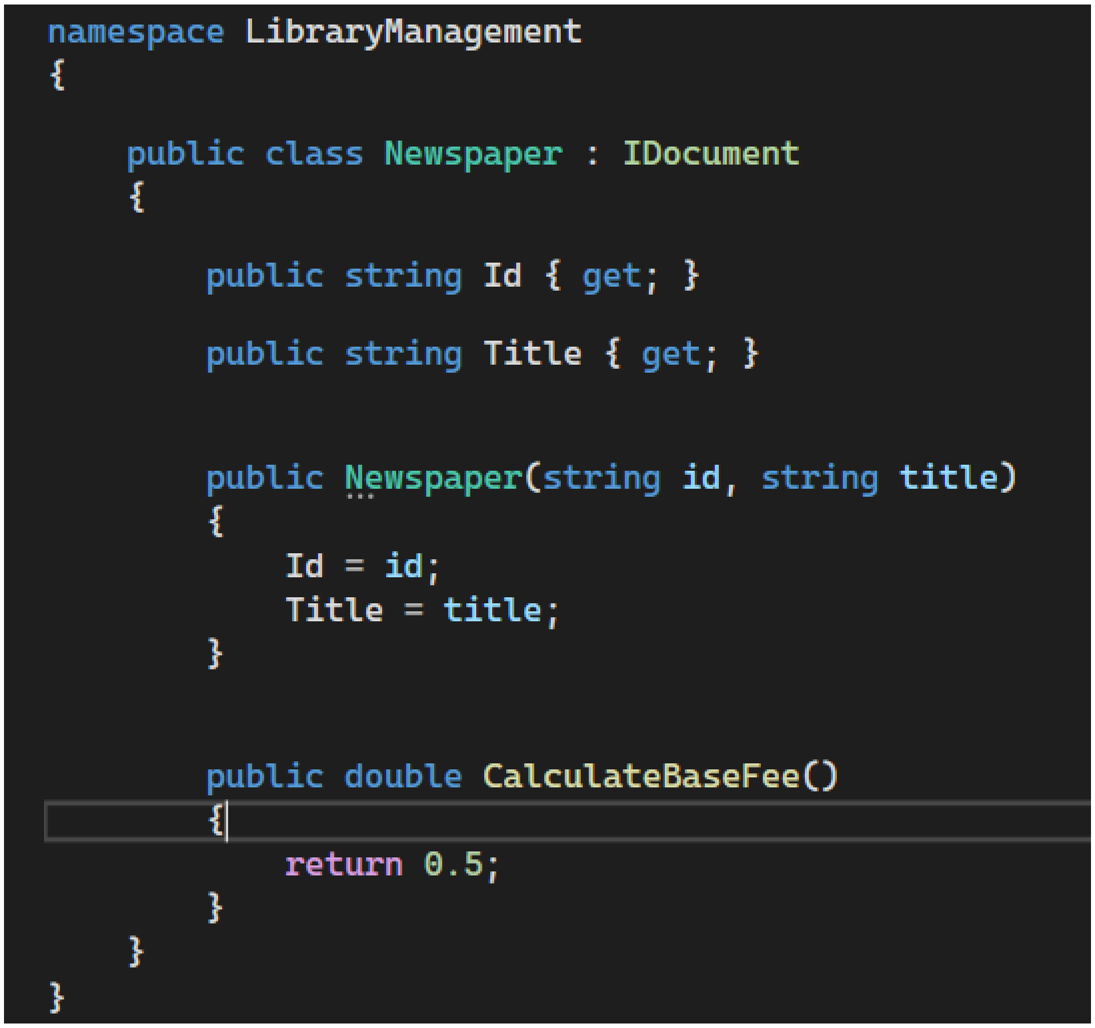
- Creates different document types (Book, Magazine, Newspaper)



```
namespace LibraryManagement
{
    16 references
    public interface IDocument
    {
        7 references
        string Id { get; }
        9 references
        string Title { get; }
        5 references
        double CalculateBaseFee();
    }
}
```

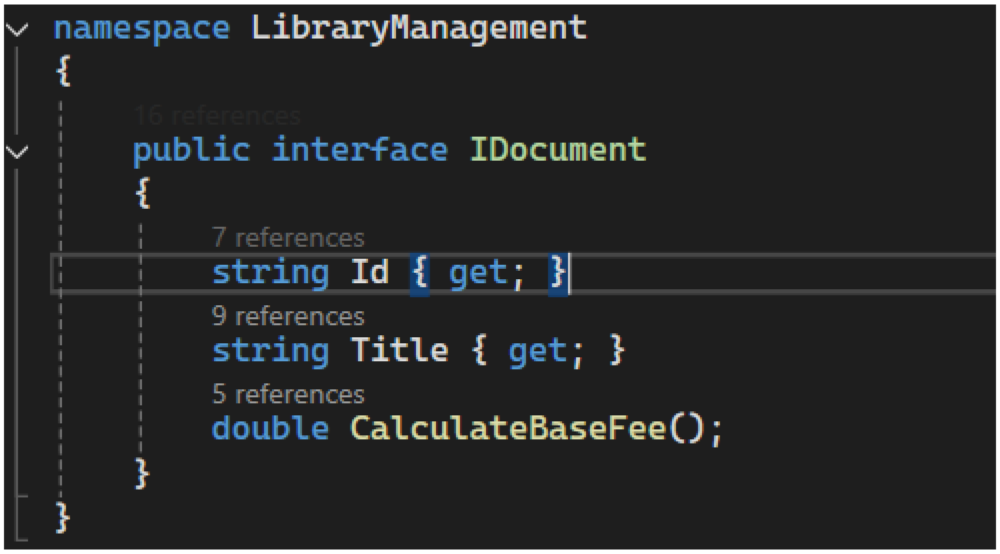


```
namespace LibraryManagement
{

    public class Magazine : IDocument
    {

        public string Id { get; }

        public string Title { get; }


        public Magazine(string id, string title)
        {
            Id = id;
            Title = title;
        }


        public double CalculateBaseFee()
        {
            return 1;
        }
    }
}
```



```
namespace LibraryManagement
{
    public class Newspaper : IDocument
    {
        public string Id { get; }
        public string Title { get; }

        public Newspaper(string id, string title)
        {
            Id = id;
            Title = title;
        }

        public double CalculateBaseFee()
        {
            return 0.5;
        }
    }
}
```

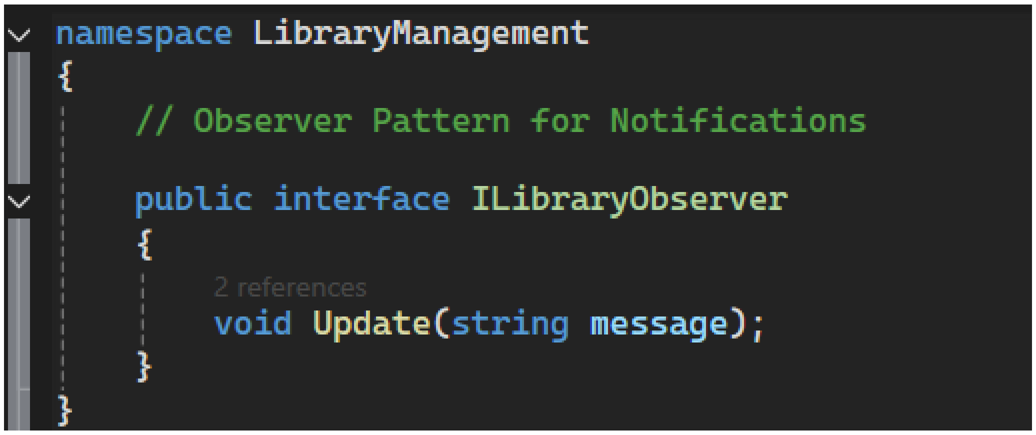
- Encapsulates object creation logic
- Easily extensible for new document types by inherit the interface below



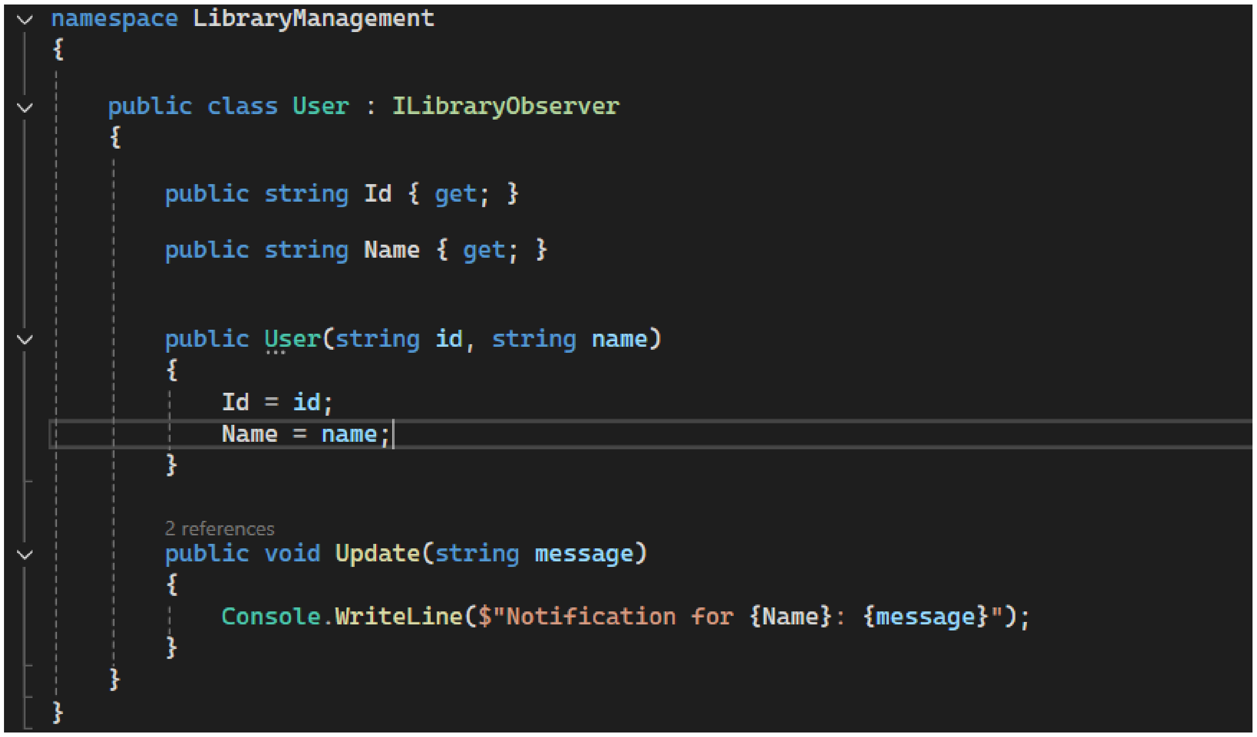
```
namespace LibraryManagement
{
    16 references
    public interface IDocument
    {
        7 references
        string Id { get; }
        9 references
        string Title { get; }
        5 references
        double CalculateBaseFee();
    }
}
```

3. Observer Pattern Application:

- Implemented with ILibraryObserver interface and User class

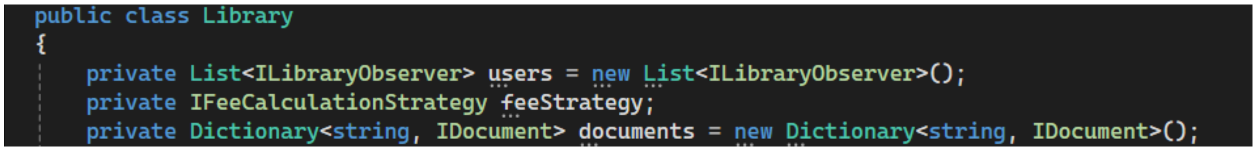


```csharp
namespace LibraryManagement
{
    // Observer Pattern for Notifications

    public interface ILibraryObserver
    {
        2 references
        void Update(string message);
    }
}
```

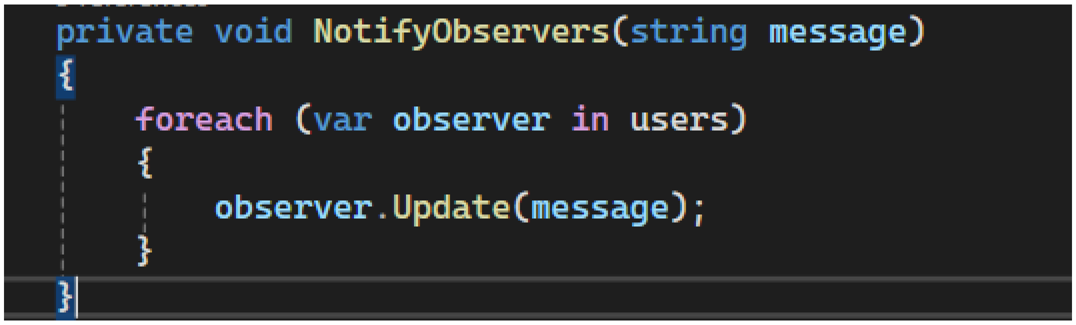


```csharp
namespace LibraryManagement
{

    public class User : ILibraryObserver
    {

        public string Id { get; }

        public string Name { get; }


        public User(string id, string name)
        {
            Id = id;
            Name = name;
        }

        2 references
        public void Update(string message)
        {
            Console.WriteLine($"Notification for {Name}: {message}");
        }
    }
}
```

- Library class maintains observer list and notification logic

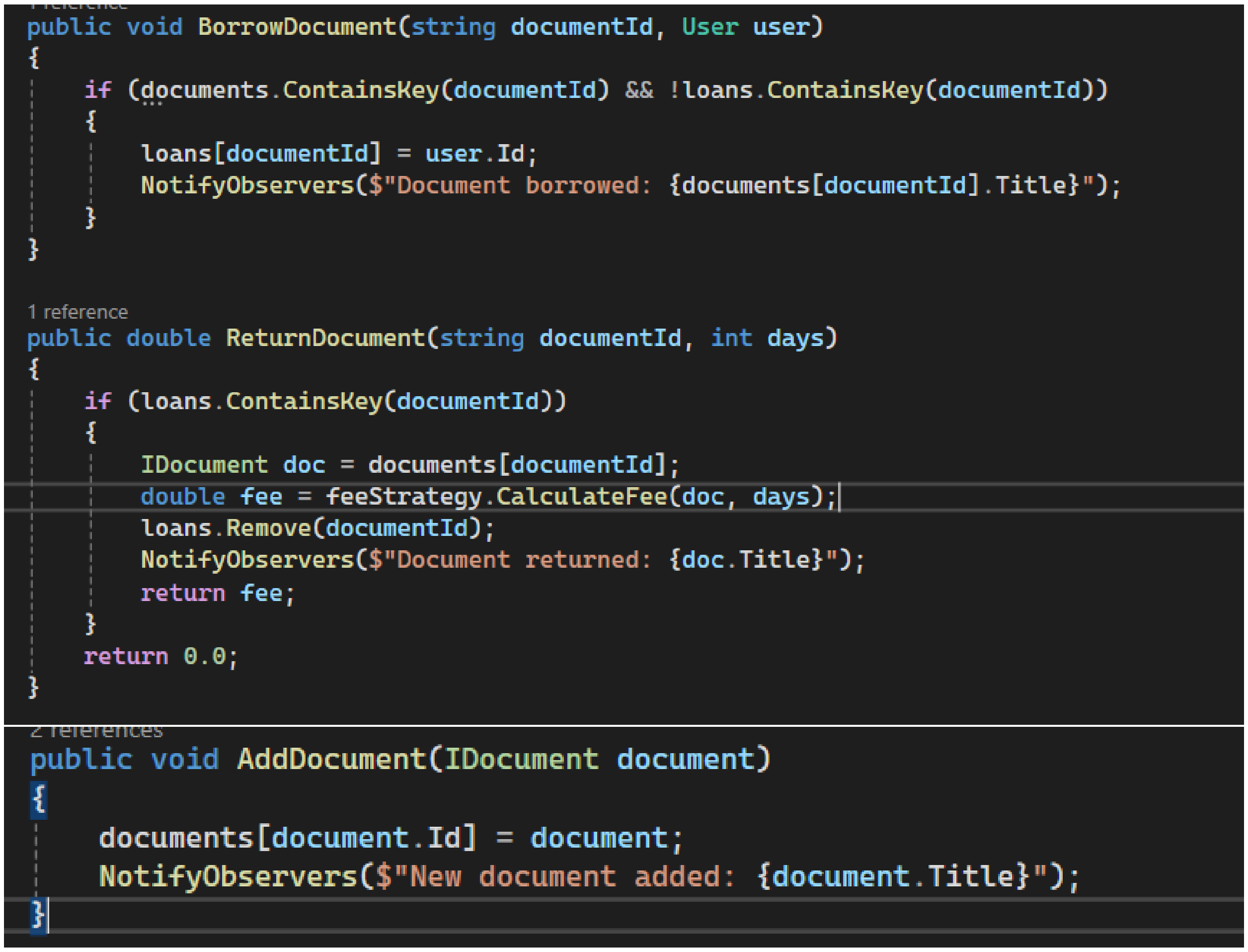


```csharp
public class Library
{
    private List<ILibraryObserver> users = new List<ILibraryObserver>();
    private IFeeCalculationStrategy feeStrategy;
    private Dictionary<string, IDocument> documents = new Dictionary<string, IDocument>();
```



```
private void NotifyObservers(string message)
{
    foreach (var observer in users)
    {
        observer.Update(message);
    }
}
```

- Notifies users about new documents and status changes



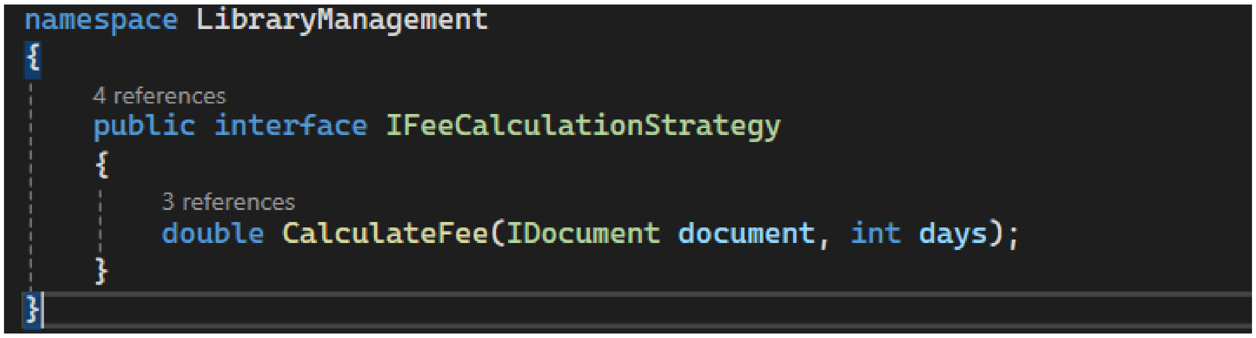
```
public void BorrowDocument(string documentId, User user)
{
    if (documents.ContainsKey(documentId) && !loans.ContainsKey(documentId))
    {
        loans[documentId] = user.Id;
        NotifyObservers($"Document borrowed: {documents[documentId].Title}");
    }
}

1 reference
public double ReturnDocument(string documentId, int days)
{
    if (loans.ContainsKey(documentId))
    {
        IDocument doc = documents[documentId];
        double fee = feeStrategy.CalculateFee(doc, days);
        loans.Remove(documentId);
        NotifyObservers($"Document returned: {doc.Title}");
        return fee;
    }
    return 0.0;
}
```

```
public void AddDocument(IDocument document)
{
    documents[document.Id] = document;
    NotifyObservers($"New document added: {document.Title}");
}
```

- Loose coupling between subjects and observers
- Supports multiple simultaneous observers

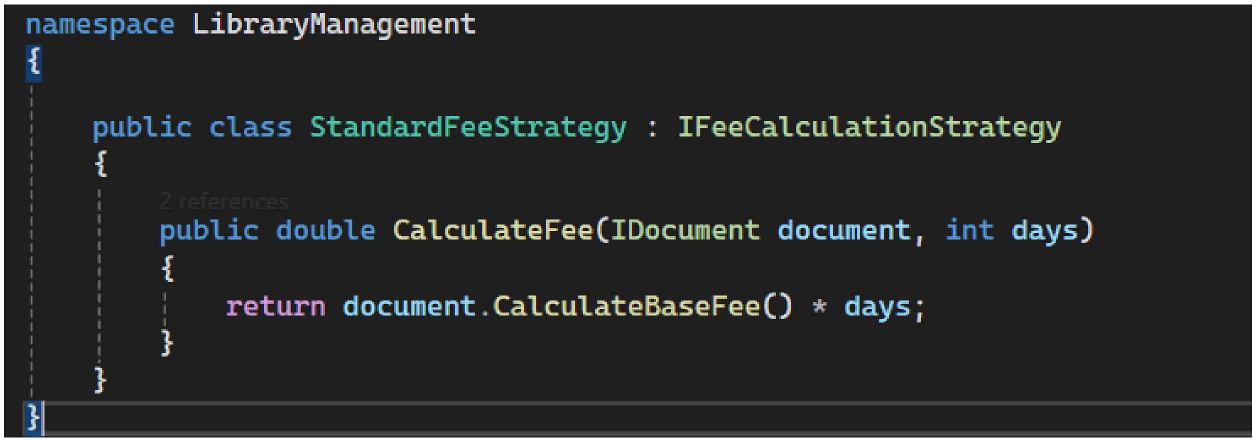
4. Strategy Pattern Application:

- Implemented with IFeeCalculationStrategy interface

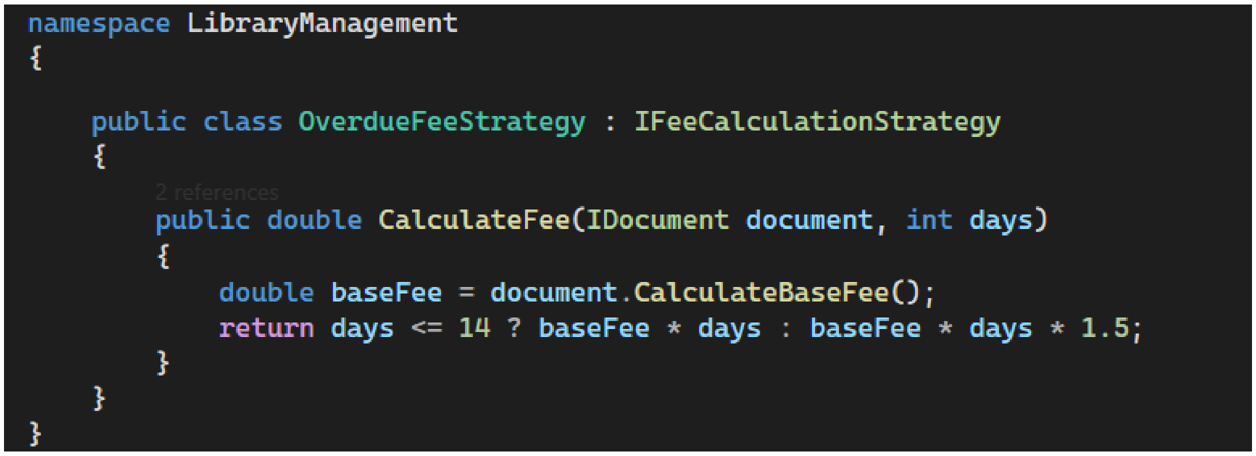


```
namespace LibraryManagement
{
    4 references
    public interface IFeeCalculationStrategy
    {
        3 references
        double CalculateFee(IDocument document, int days);
    }
}
```

- Concrete strategies: StandardFeeStrategy and OverdueFeeStrategy

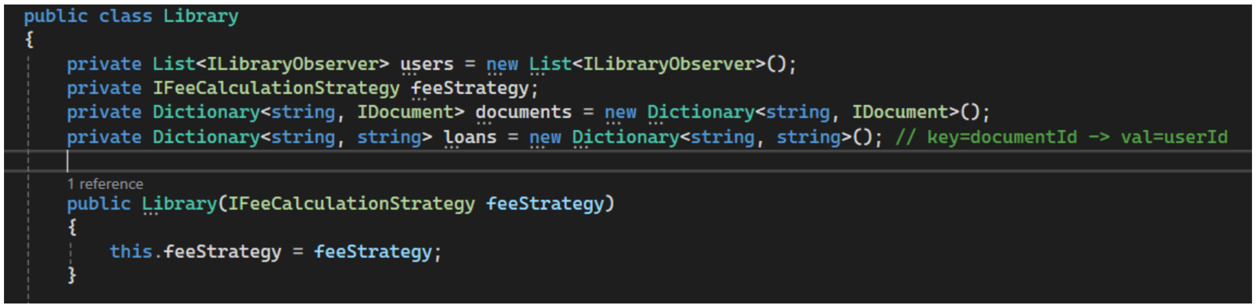


```
namespace LibraryManagement
{

    public class StandardFeeStrategy : IFeeCalculationStrategy
    {
        2 references
        public double CalculateFee(IDocument document, int days)
        {
            return document.CalculateBaseFee() * days;
        }
    }
}
```



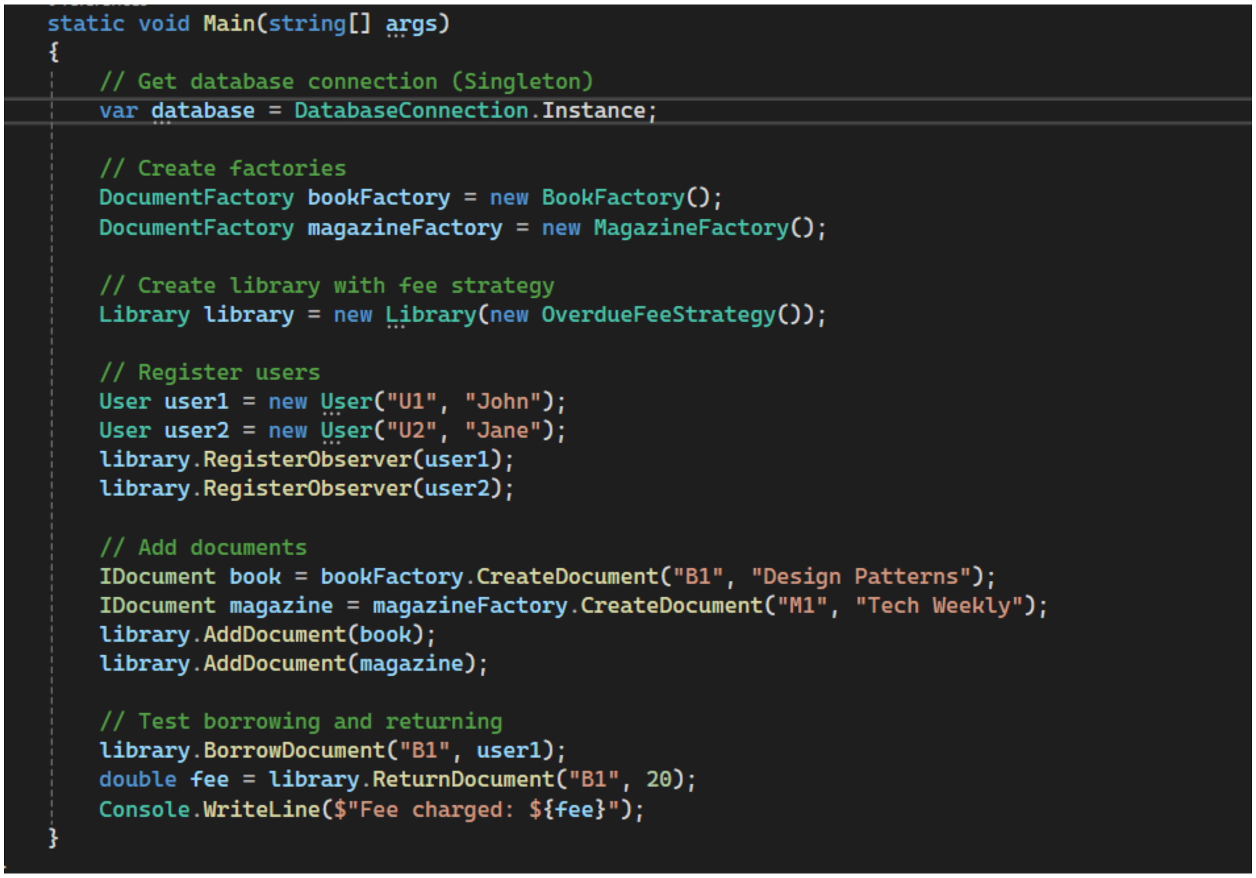
```
namespace LibraryManagement
{

    public class OverdueFeeStrategy : IFeeCalculationStrategy
    {
        2 references
        public double CalculateFee(IDocument document, int days)
        {
            double baseFee = document.CalculateBaseFee();
            return days <= 14 ? baseFee * days : baseFee * days * 1.5;
        }
    }
}
```

- Calculates fees based on document type and duration
- Allows runtime strategy changes
- Encapsulates fee calculation algorithms



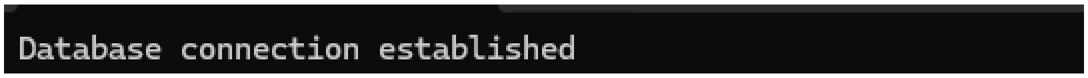
```csharp
public class Library
{
    private List<ILibraryObserver> users = new List<ILibraryObserver>();
    private IFeeCalculationStrategy feeStrategy;
    private Dictionary<string, IDocument> documents = new Dictionary<string, IDocument>();
    private Dictionary<string, string> loans = new Dictionary<string, string>(); // key=documentId -> val=userId

    1 reference
    public Library(IFeeCalculationStrategy feeStrategy)
    {
        this.feeStrategy = feeStrategy;
    }
```

II. Demo code:



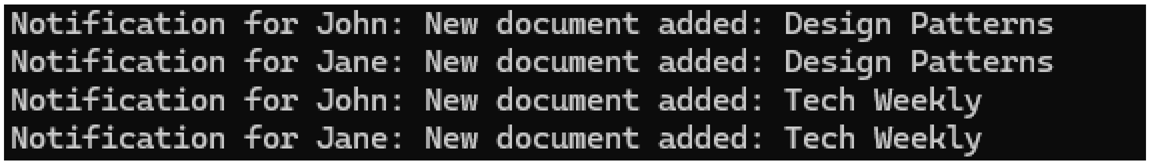
```csharp
static void Main(string[] args)
{
    // Get database connection (Singleton)
    var database = DatabaseConnection.Instance;

    // Create factories
    DocumentFactory bookFactory = new BookFactory();
    DocumentFactory magazineFactory = new MagazineFactory();

    // Create library with fee strategy
    Library library = new Library(new OverdueFeeStrategy());

    // Register users
    User user1 = new User("U1", "John");
    User user2 = new User("U2", "Jane");
    library.RegisterObserver(user1);
    library.RegisterObserver(user2);

    // Add documents
    IDocument book = bookFactory.CreateDocument("B1", "Design Patterns");
    IDocument magazine = magazineFactory.CreateDocument("M1", "Tech Weekly");
    library.AddDocument(book);
    library.AddDocument(magazine);

    // Test borrowing and returning
    library.BorrowDocument("B1", user1);
    double fee = library.ReturnDocument("B1", 20);
    Console.WriteLine($"Fee charged: ${fee}");
}
```

1. Ensure only one database is connected:



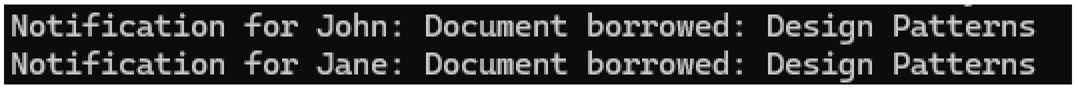
```
Database connection established
```

2. Add new books and all users will be informed:



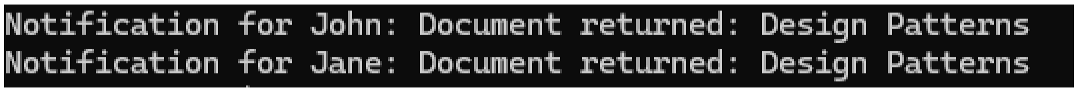
```
Notification for John: New document added: Design Patterns
Notification for Jane: New document added: Design Patterns
Notification for John: New document added: Tech Weekly
Notification for Jane: New document added: Tech Weekly
```

3. Borrow books and all users will be informed:



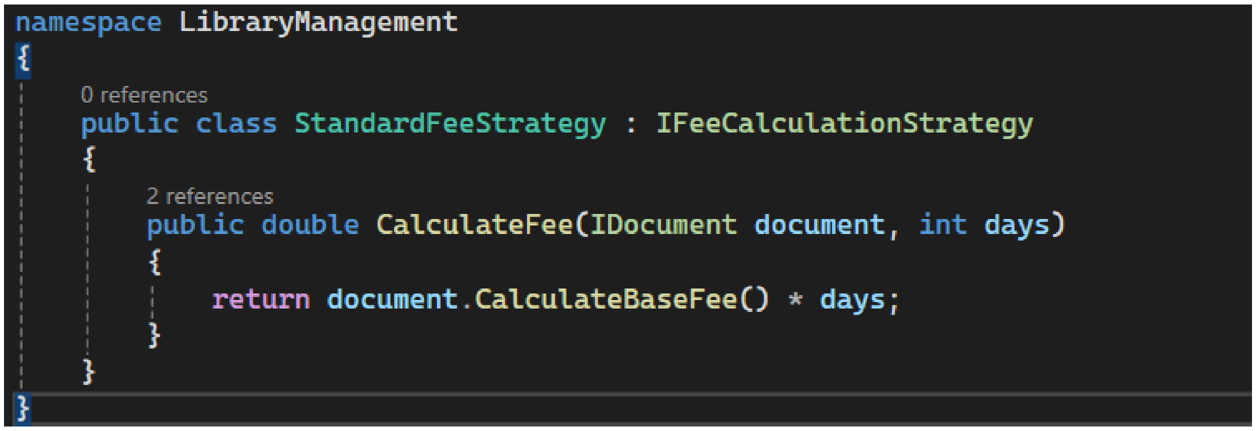
```
Notification for John: Document borrowed: Design Patterns
Notification for Jane: Document borrowed: Design Patterns
```

4. Return books and all users will be informed:

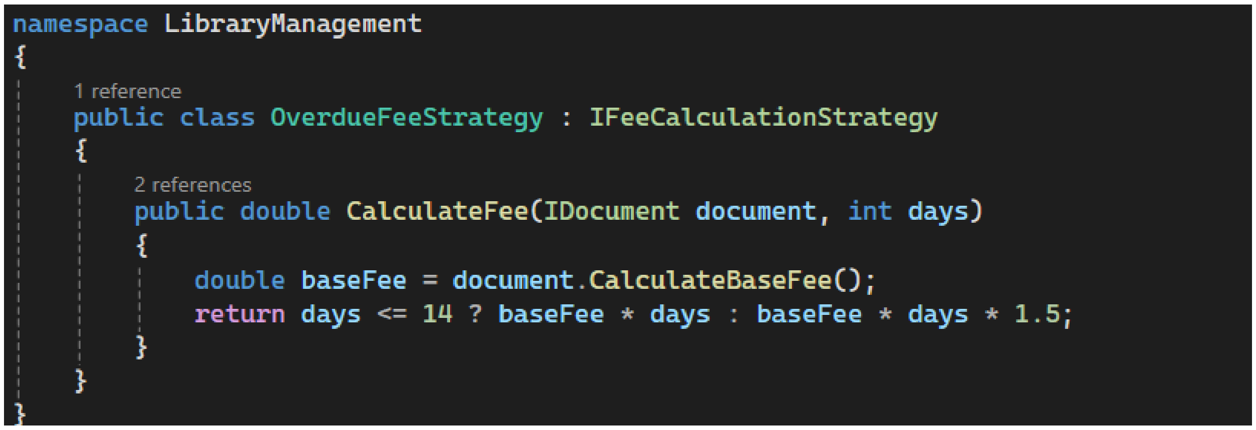


```
Notification for John: Document returned: Design Patterns
Notification for Jane: Document returned: Design Patterns
```

5. Algorithm to calculate borrow fee:

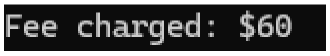


```
namespace LibraryManagement
{
    0 references
    public class StandardFeeStrategy : IFeeCalculationStrategy
    {
        2 references
        public double CalculateFee(IDocument document, int days)
        {
            return document.CalculateBaseFee() * days;
        }
    }
}
```



```
namespace LibraryManagement
{
    1 reference
    public class OverdueFeeStrategy : IFeeCalculationStrategy
    {
        2 references
        public double CalculateFee(IDocument document, int days)
        {
            double baseFee = document.CalculateBaseFee();
            return days <= 14 ? baseFee * days : baseFee * days * 1.5;
        }
    }
}
```

And the result with OverdueFeeStrategy (overdue 20 days):



```
Fee charged: $60
```