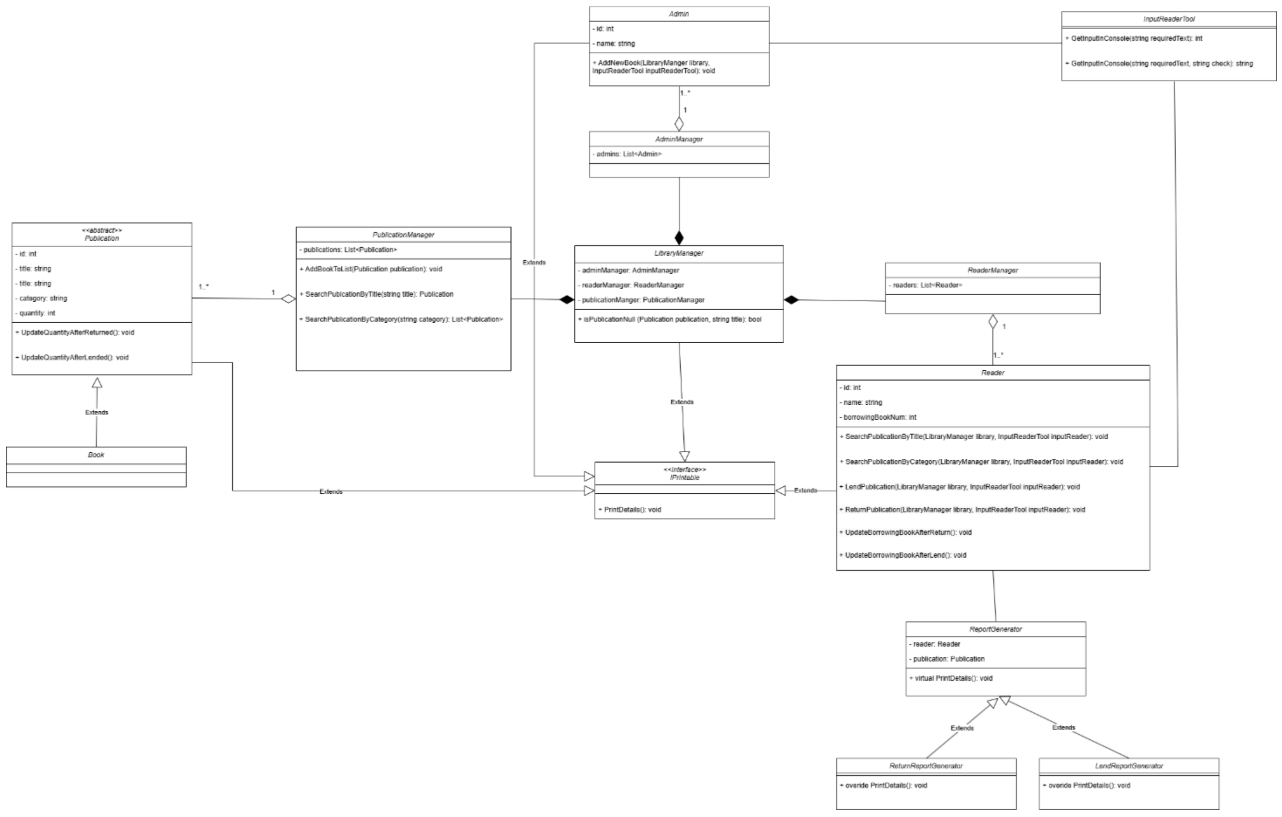
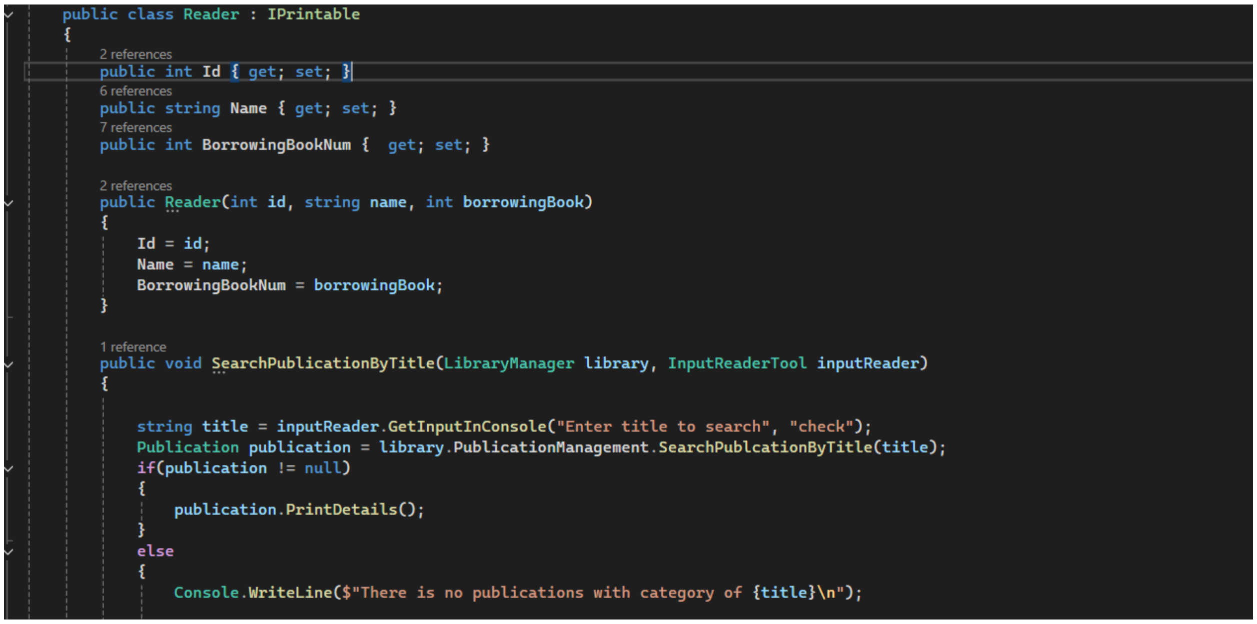
Lab 4 – Report – SOLID in Library Management System

I. Class Diagram.

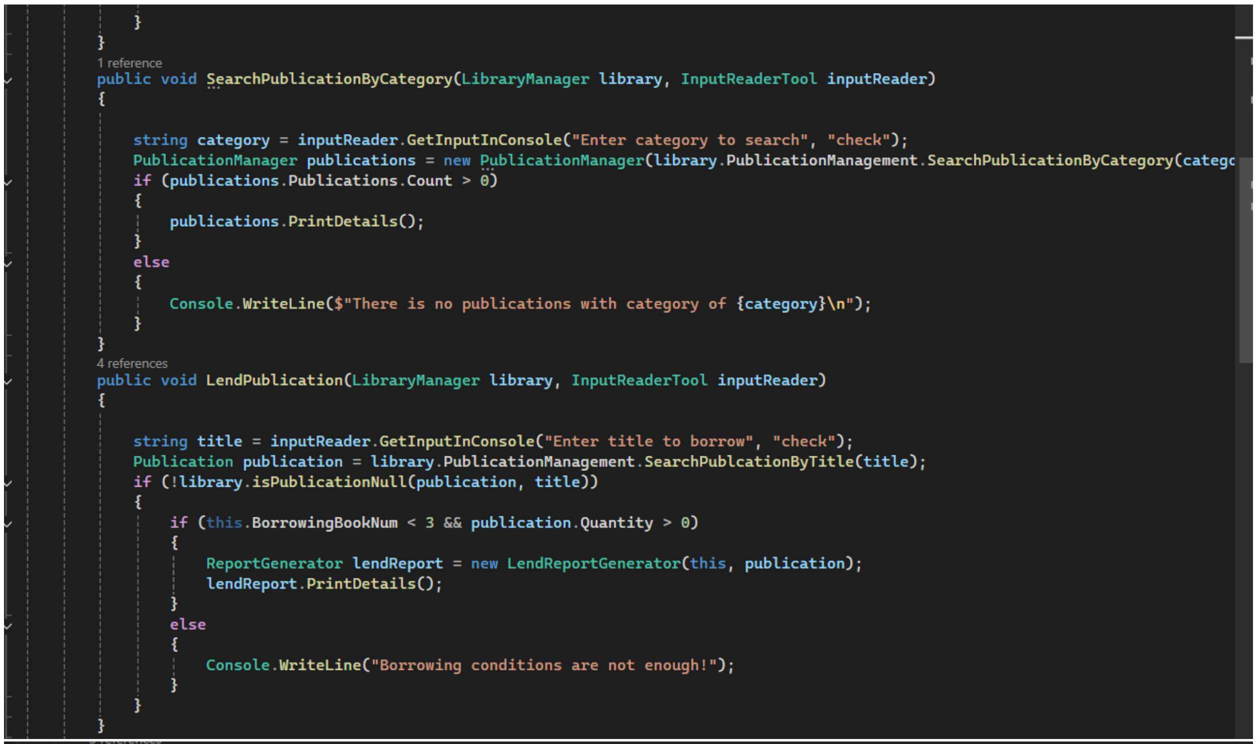

Admin
- id: int
- name: string
+ AddNewBook(LibraryManger library, InputReaderTool inputReaderTool): void
InputReaderTool
+ GetInputInConsole(string requiredText): int
+ GetInputInConsole(string requiredText, string check): string
1..*
1
AdminManager
- admins: List<Admin>
<<abstract>> Publication
- id: int
- title: string
- title: string
- category: string
- quantity: int
+ UpdateQuantityAfterReturned(): void
+ UpdateQuantityAfterLended(): void
1..*
1
PublicationManager
- publications: List<Publication>
+ AddBookToList(Publication publication): void
+ SearchPublicationByTitle(string title): Publication
+ SearchPublicationByCategory(string category): List<Publication>
Extends
LibraryManager
- adminManager: AdminManager
- readerManager: ReaderManager
- publicationManger: PublicationManager
+ isPublicationNull (Publication publication, string title): bool
ReaderManager
- readers: List<Reader>
1
1..*
Reader
- id: int
- name: string
- borrowingBookNum: int
+ SearchPublicationByTitle(LibraryManager library, InputReaderTool inputReader): void
+ SearchPublicationByCategory(LibraryManager library, InputReaderTool inputReader): void
+ LendPublication(LibraryManager library, InputReaderTool inputReader): void
+ ReturnPublication(LibraryManager library, InputReaderTool inputReader): void
+ UpdateBorrowingBookAfterReturn(): void
+ UpdateBorrowingBookAfterLend(): void
Extends
Book
Extends
Extends
<<interface>> IPrintable
+ PrintDetails(): void
Extends
ReportGenerator
- reader: Reader
- publication: Publication
+ virtual PrintDetails(): void
Extends
Extends
ReturnReportGenerator
+ overide PrintDetails(): void
LendReportGenerator
+ overide PrintDetails(): void

II. Explain SOLID.

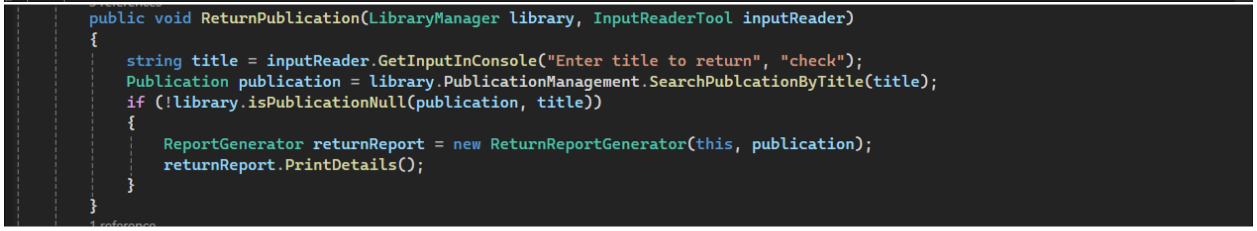
1. Single Responsibility Principle: Each object has vital functionalities that it should have.



```
public class Reader : IPrintable
{
    2 references
    public int Id { get; set; }
    6 references
    public string Name { get; set; }
    7 references
    public int BorrowingBookNum {  get; set; }

    2 references
    public Reader(int id, string name, int borrowingBook)
    {
        Id = id;
        Name = name;
        BorrowingBookNum = borrowingBook;
    }

    1 reference
    public void SearchPublicationByTitle(LibraryManager library, InputReaderTool inputReader)
    {

        string title = inputReader.GetInputInConsole("Enter title to search", "check");
        Publication publication = library.PublicationManagement.SearchPublcationByTitle(title);
        if(publication != null)
        {
            publication.PrintDetails();
        }
        else
        {
            Console.WriteLine($"There is no publications with category of {title}\n");
```

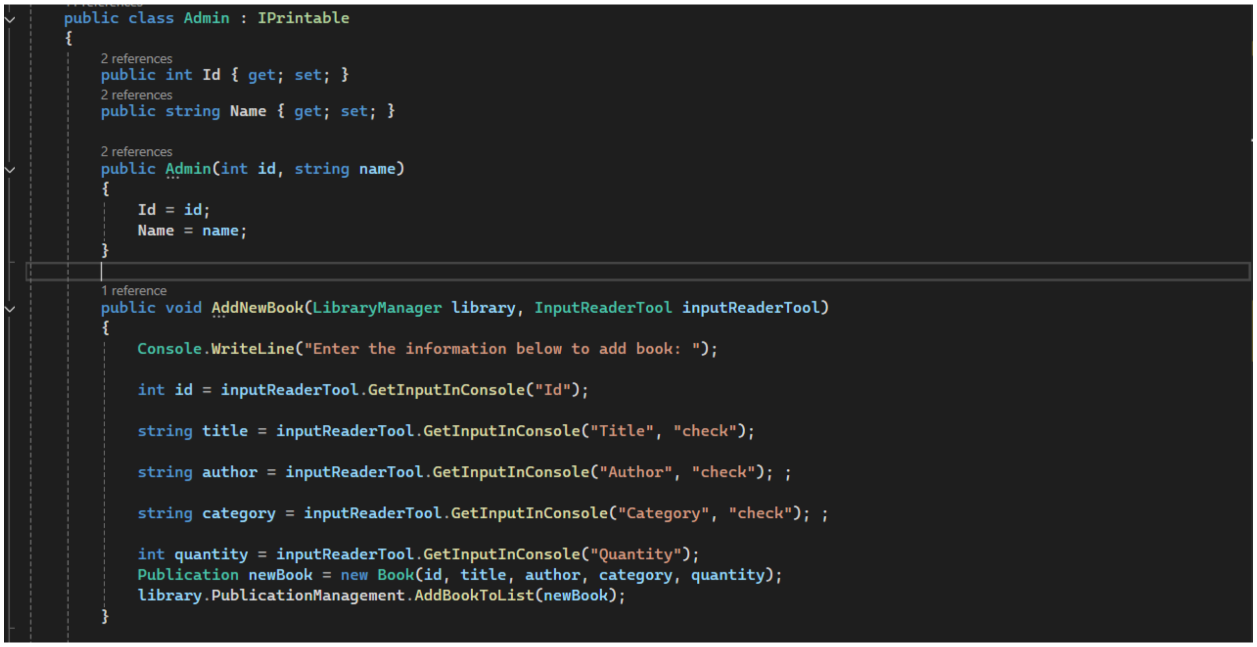


```
            }
        }
        1 reference
        public void SearchPublicationByCategory(LibraryManager library, InputReaderTool inputReader)
        {

            string category = inputReader.GetInputInConsole("Enter category to search", "check");
            PublicationManager publications = new PublicationManager(library.PublicationManagement.SearchPublicationByCategory(catego
            if (publications.Publications.Count > 0)
            {
                publications.PrintDetails();
            }
            else
            {
                Console.WriteLine($"There is no publications with category of {category}\n");
            }
        }
        4 references
        public void LendPublication(LibraryManager library, InputReaderTool inputReader)
        {

            string title = inputReader.GetInputInConsole("Enter title to borrow", "check");
            Publication publication = library.PublicationManagement.SearchPublcationByTitle(title);
            if (!library.isPublicationNull(publication, title))
            {
                if (this.BorrowingBookNum < 3 && publication.Quantity > 0)
                {
                    ReportGenerator lendReport = new LendReportGenerator(this, publication);
                    lendReport.PrintDetails();
                }
                else
                {
                    Console.WriteLine("Borrowing conditions are not enough!");
                }
            }
        }
```



```
        public void ReturnPublication(LibraryManager library, InputReaderTool inputReader)
        {
            string title = inputReader.GetInputInConsole("Enter title to return", "check");
            Publication publication = library.PublicationManagement.SearchPublcationByTitle(title);
            if (!library.isPublicationNull(publication, title))
            {
                ReportGenerator returnReport = new ReturnReportGenerator(this, publication);
                returnReport.PrintDetails();
            }
        }
```

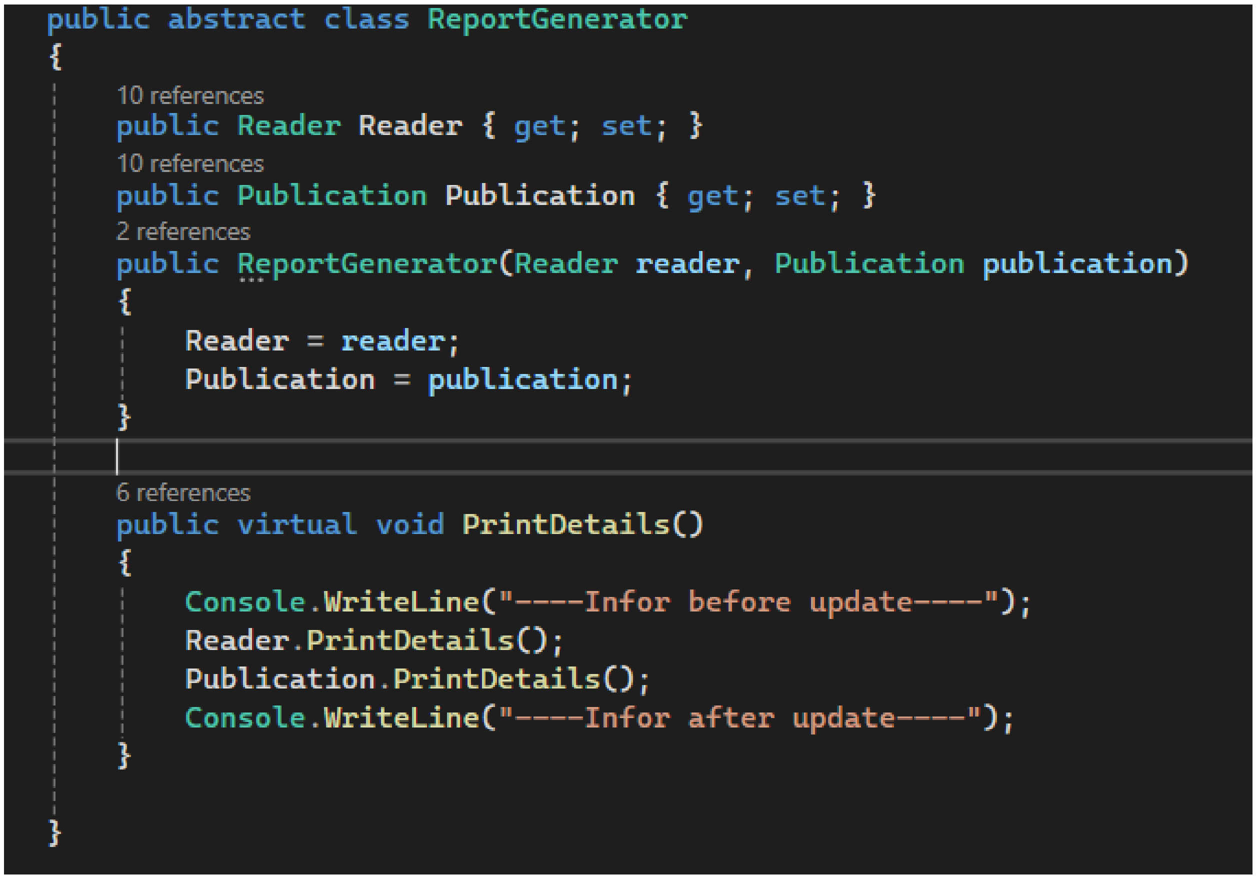
Reader has simple functionalities such as: search for a book, lend books and return book.



```
    public class Admin : IPrintable
    {
        2 references
        public int Id { get; set; }
        2 references
        public string Name { get; set; }

        2 references
        public Admin(int id, string name)
        {
            Id = id;
            Name = name;
        }

        1 reference
        public void AddNewBook(LibraryManager library, InputReaderTool inputReaderTool)
        {
            Console.WriteLine("Enter the information below to add book: ");

            int id = inputReaderTool.GetInputInConsole("Id");

            string title = inputReaderTool.GetInputInConsole("Title", "check");

            string author = inputReaderTool.GetInputInConsole("Author", "check"); ;

            string category = inputReaderTool.GetInputInConsole("Category", "check"); ;

            int quantity = inputReaderTool.GetInputInConsole("Quantity");
            Publication newBook = new Book(id, title, author, category, quantity);
            library.PublicationManagement.AddBookToList(newBook);
        }
```

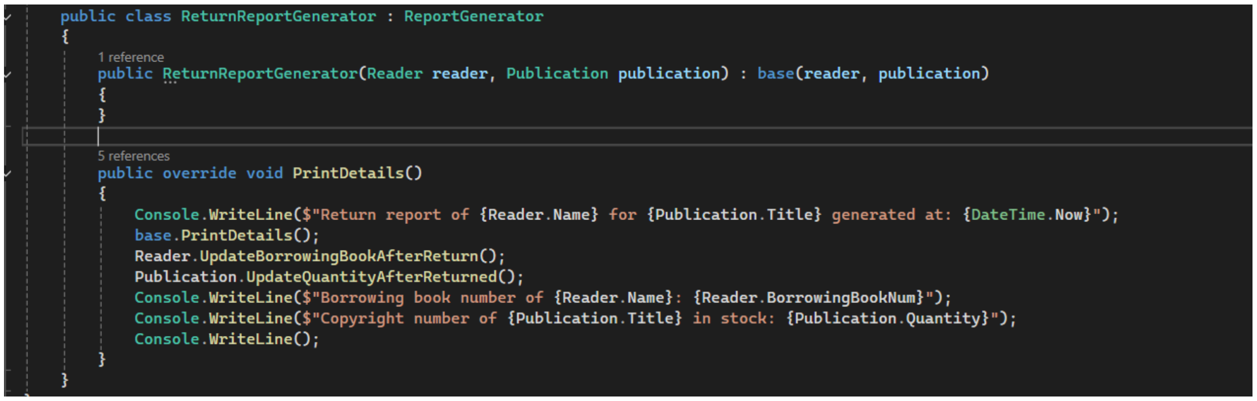
Admin object has simple functionalities such as: add new books into the library.

2. Open/Closed Principle: we can extend a class's behavior without modifying it.
   I create a abstract class for ReportGenerator so that, we can generate different types of reports later.



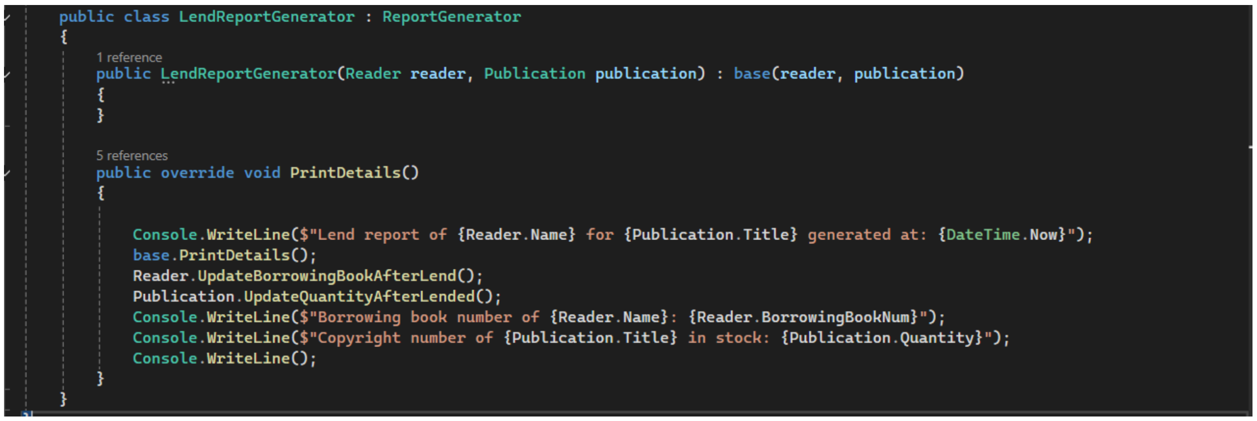
```
public abstract class ReportGenerator
{
    10 references
    public Reader Reader { get; set; }
    10 references
    public Publication Publication { get; set; }
    2 references
    public ReportGenerator(Reader reader, Publication publication)
    {
        Reader = reader;
        Publication = publication;
    }

    6 references
    public virtual void PrintDetails()
    {
        Console.WriteLine("----Infor before update----");
        Reader.PrintDetails();
        Publication.PrintDetails();
        Console.WriteLine("----Infor after update----");
    }

}
```

This ReturnReportGenerator extends the ReportGenerator to so information when readers return books:



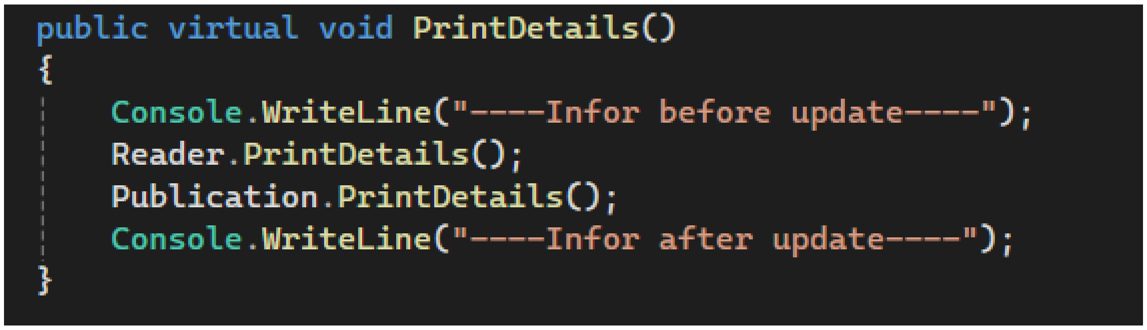
```
public class ReturnReportGenerator : ReportGenerator
{
    1 reference
    public ReturnReportGenerator(Reader reader, Publication publication) : base(reader, publication)
    {
    }

    5 references
    public override void PrintDetails()
    {
        Console.WriteLine($"Return report of {Reader.Name} for {Publication.Title} generated at: {DateTime.Now}");
        base.PrintDetails();
        Reader.UpdateBorrowingBookAfterReturn();
        Publication.UpdateQuantityAfterReturned();
        Console.WriteLine($"Borrowing book number of {Reader.Name}: {Reader.BorrowingBookNum}");
        Console.WriteLine($"Copyright number of {Publication.Title} in stock: {Publication.Quantity}");
        Console.WriteLine();
    }
}
```

This LendReportGenerator extends the ReportGenerator to so information when readers lend books:



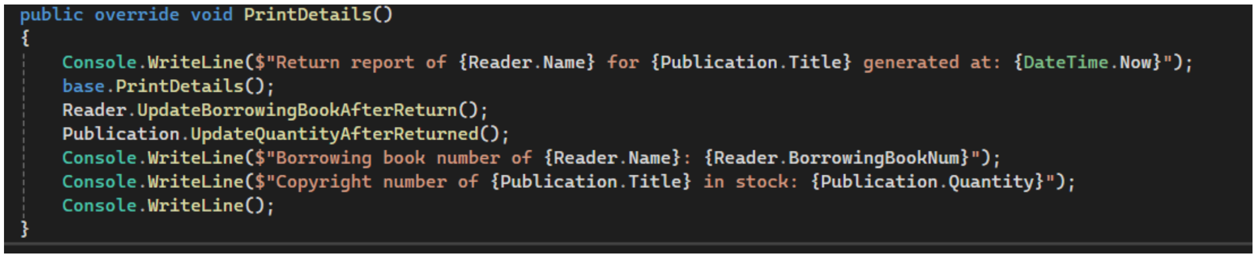
```
public class LendReportGenerator : ReportGenerator
{
    1 reference
    public LendReportGenerator(Reader reader, Publication publication) : base(reader, publication)
    {
    }

    5 references
    public override void PrintDetails()
    {

        Console.WriteLine($"Lend report of {Reader.Name} for {Publication.Title} generated at: {DateTime.Now}");
        base.PrintDetails();
        Reader.UpdateBorrowingBookAfterLend();
        Publication.UpdateQuantityAfterLended();
        Console.WriteLine($"Borrowing book number of {Reader.Name}: {Reader.BorrowingBookNum}");
        Console.WriteLine($"Copyright number of {Publication.Title} in stock: {Publication.Quantity}");
        Console.WriteLine();
    }
}
```

3. Liskov Substitution Principle: my subclass can utilize the superclass without causing problems
   Superclass:



```
public virtual void PrintDetails()
{
    Console.WriteLine("----Infor before update----");
    Reader.PrintDetails();
    Publication.PrintDetails();
    Console.WriteLine("----Infor after update----");
}
```

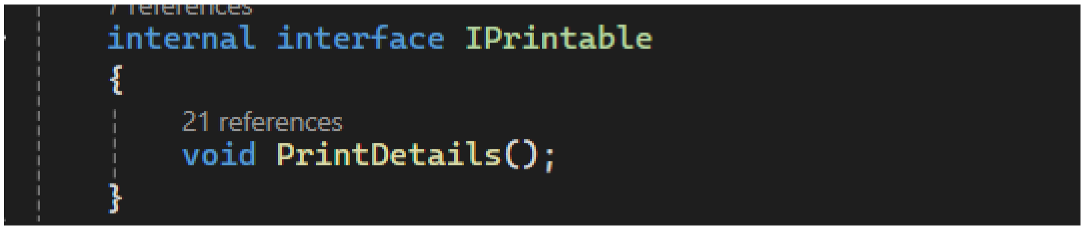
   Subclass:



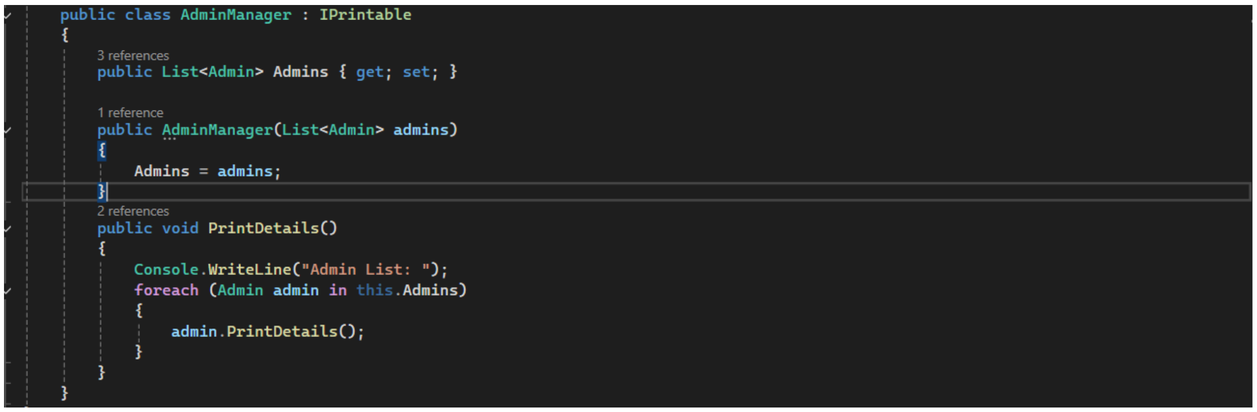
```
public override void PrintDetails()
{
    Console.WriteLine($"Return report of {Reader.Name} for {Publication.Title} generated at: {DateTime.Now}");
    base.PrintDetails();
    Reader.UpdateBorrowingBookAfterReturn();
    Publication.UpdateQuantityAfterReturned();
    Console.WriteLine($"Borrowing book number of {Reader.Name}: {Reader.BorrowingBookNum}");
    Console.WriteLine($"Copyright number of {Publication.Title} in stock: {Publication.Quantity}");
    Console.WriteLine();
}
```

   I utilize the superclass to minimum coding lines the subclass instead of changing the superclass to serve my subclass.

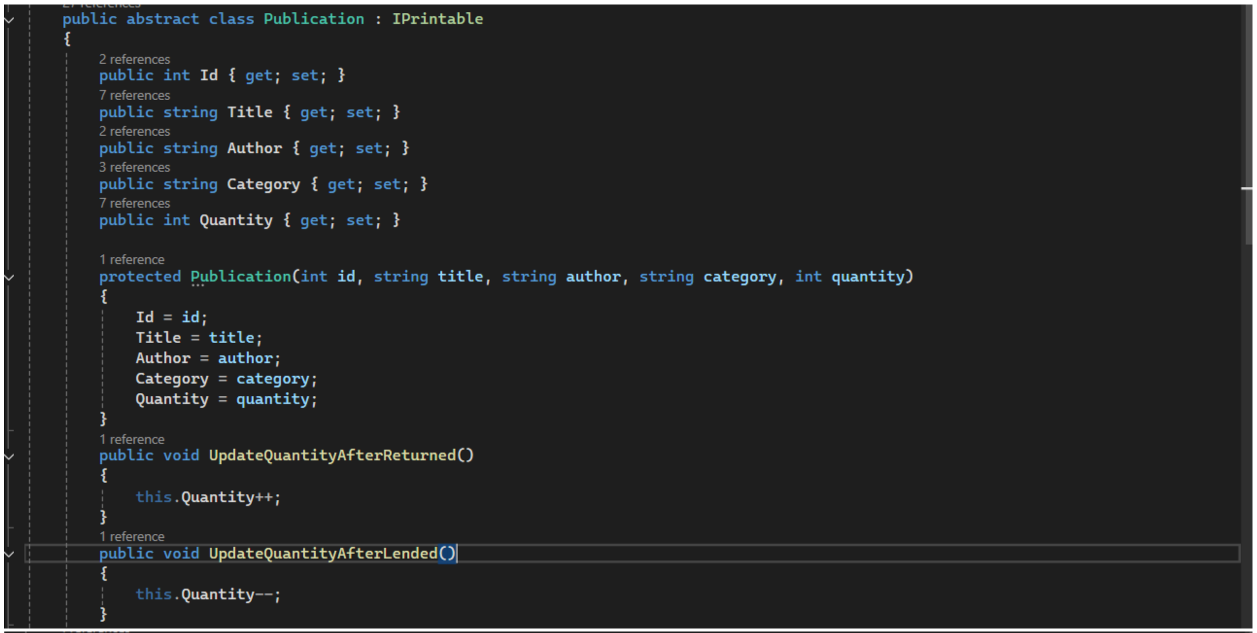
4. Interface Segregation Principle: I generate Interface to serve my implemented class, so that it do not create any redundant factors in my subclass:

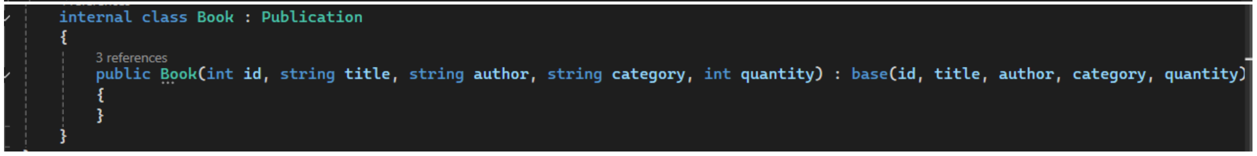


```
7 references
internal interface IPrintable
{
    21 references
    void PrintDetails();
}
```


```
public class AdminManager : IPrintable
{
    3 references
    public List<Admin> Admins { get; set; }

    1 reference
    public AdminManager(List<Admin> admins)
    {
        Admins = admins;
    }
    2 references
    public void PrintDetails()
    {
        Console.WriteLine("Admin List: ");
        foreach (Admin admin in this.Admins)
        {
            admin.PrintDetails();
        }
    }
}
```

5. Dependency Inversion Principle:
High-level modules should not depend on low-level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.

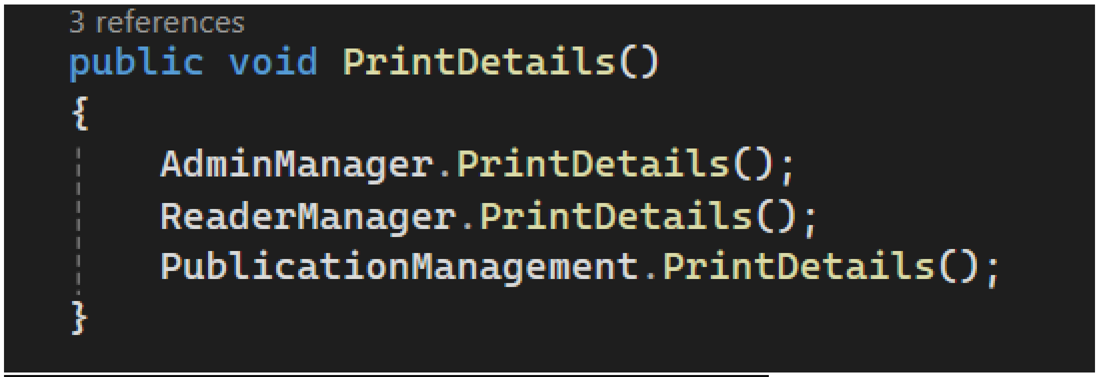

```
public abstract class Publication : IPrintable
{
    2 references
    public int Id { get; set; }
    7 references
    public string Title { get; set; }
    2 references
    public string Author { get; set; }
    3 references
    public string Category { get; set; }
    7 references
    public int Quantity { get; set; }

    1 reference
    protected Publication(int id, string title, string author, string category, int quantity)
    {
        Id = id;
        Title = title;
        Author = author;
        Category = category;
        Quantity = quantity;
    }
    1 reference
    public void UpdateQuantityAfterReturned()
    {
        this.Quantity++;
    }
    1 reference
    public void UpdateQuantityAfterLended()
    {
        this.Quantity--;
    }
```


```
internal class Book : Publication
{
    3 references
    public Book(int id, string title, string author, string category, int quantity) : base(id, title, author, category, quantity)
    {
    }
}
```

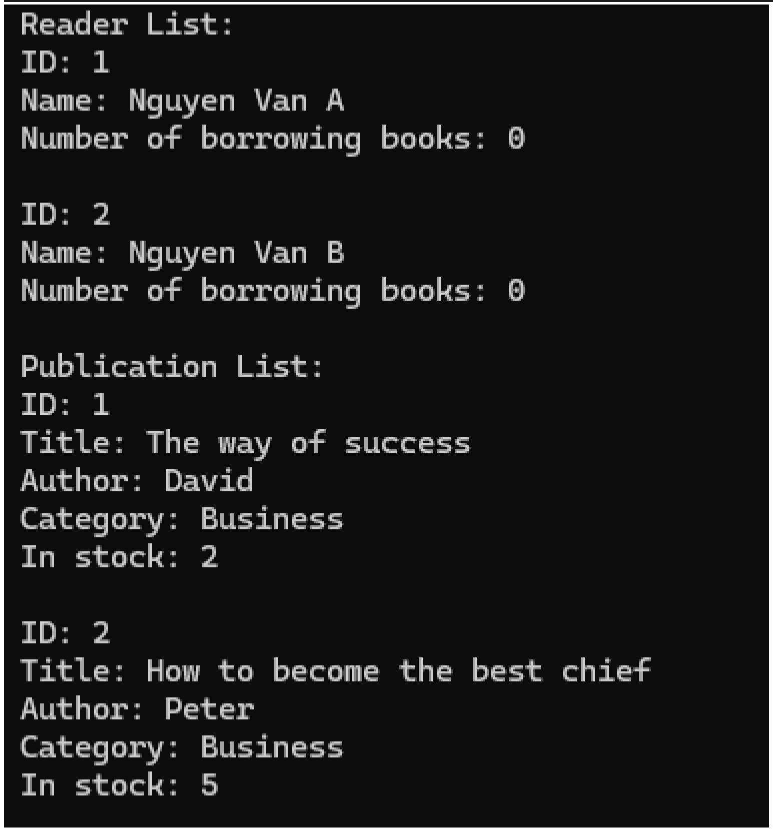
By doing this I can generate different types of publication later.

III. Demo functionalities.

Library can show all of fields:

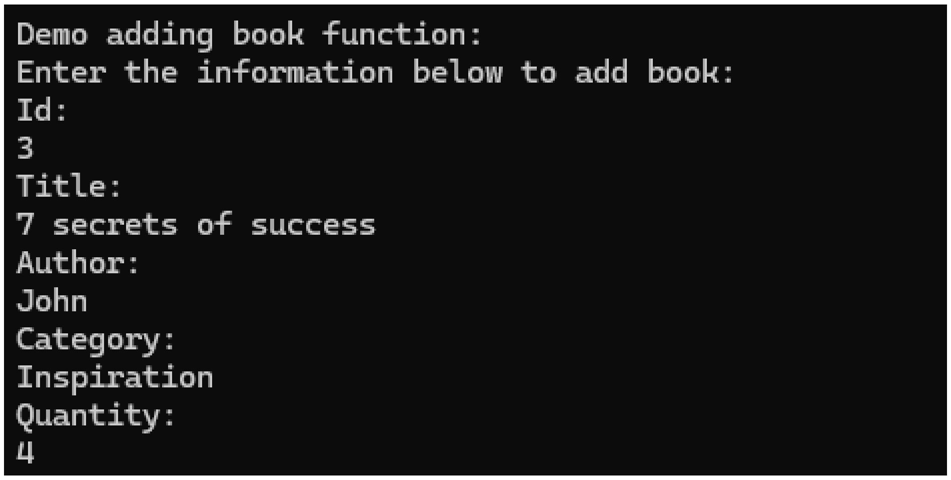


```
3 references
public void PrintDetails()
{
    AdminManager.PrintDetails();
    ReaderManager.PrintDetails();
    PublicationManagement.PrintDetails();
}
```

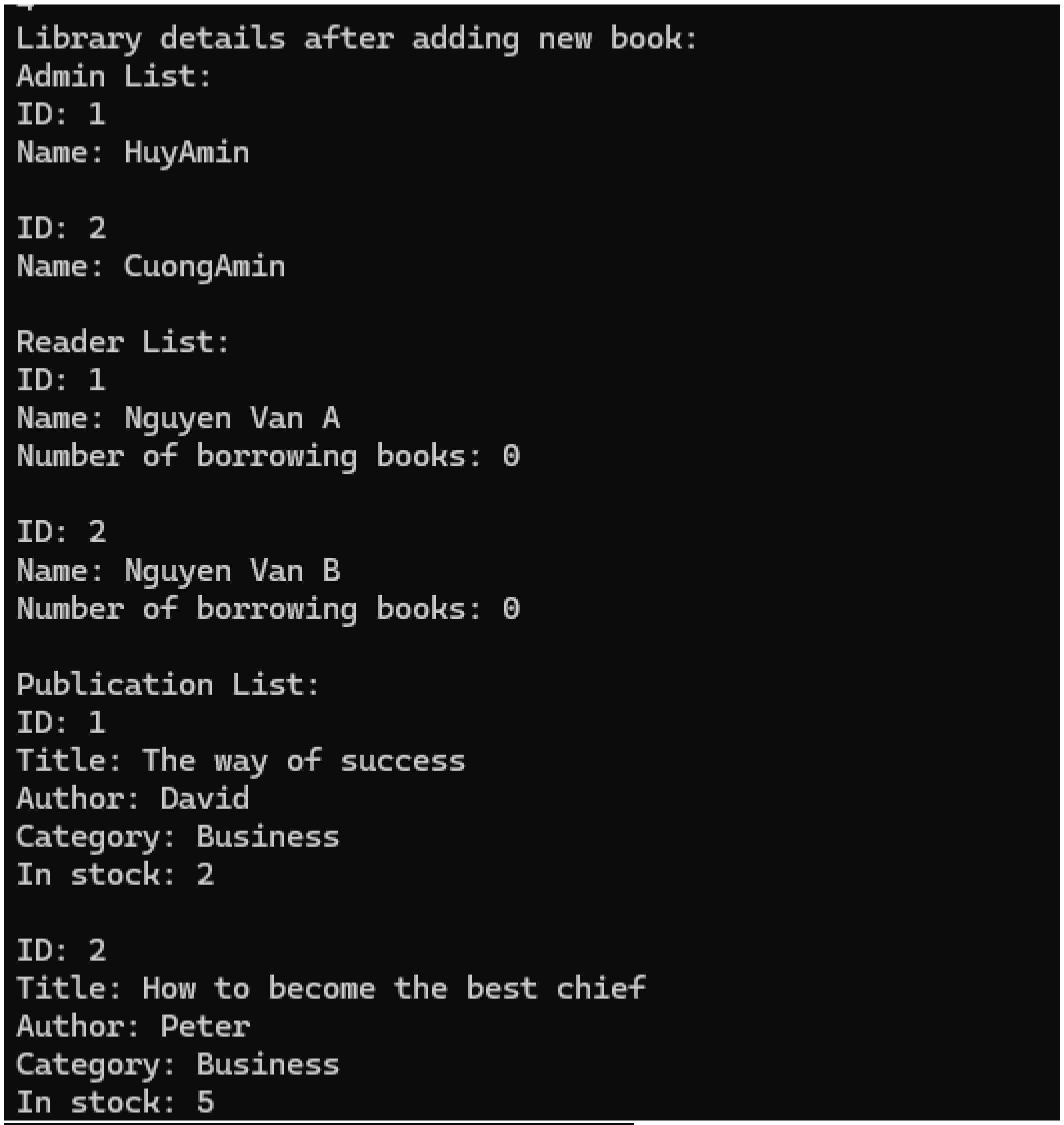


```
Reader List:
ID: 1
Name: Nguyen Van A
Number of borrowing books: 0

ID: 2
Name: Nguyen Van B
Number of borrowing books: 0

Publication List:
ID: 1
Title: The way of success
Author: David
Category: Business
In stock: 2

ID: 2
Title: How to become the best chief
Author: Peter
Category: Business
In stock: 5
```

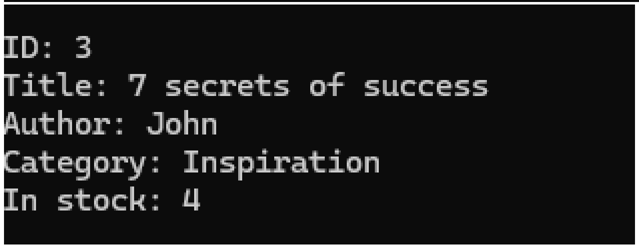
Admin can add new book into the system:



```
Demo adding book function:
Enter the information below to add book:
Id:
3
Title:
7 secrets of success
Author:
John
Category:
Inspiration
Quantity:
4
```

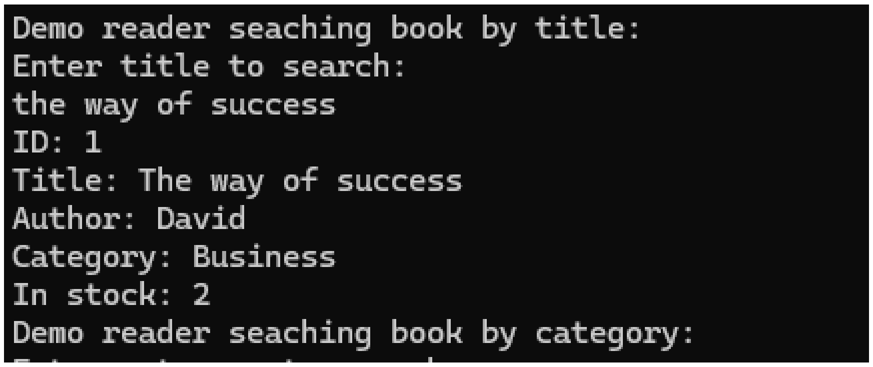


```
Library details after adding new book:
Admin List:
ID: 1
Name: HuyAmin

ID: 2
Name: CuongAmin

Reader List:
ID: 1
Name: Nguyen Van A
Number of borrowing books: 0

ID: 2
Name: Nguyen Van B
Number of borrowing books: 0

Publication List:
ID: 1
Title: The way of success
Author: David
Category: Business
In stock: 2

ID: 2
Title: How to become the best chief
Author: Peter
Category: Business
In stock: 5
```



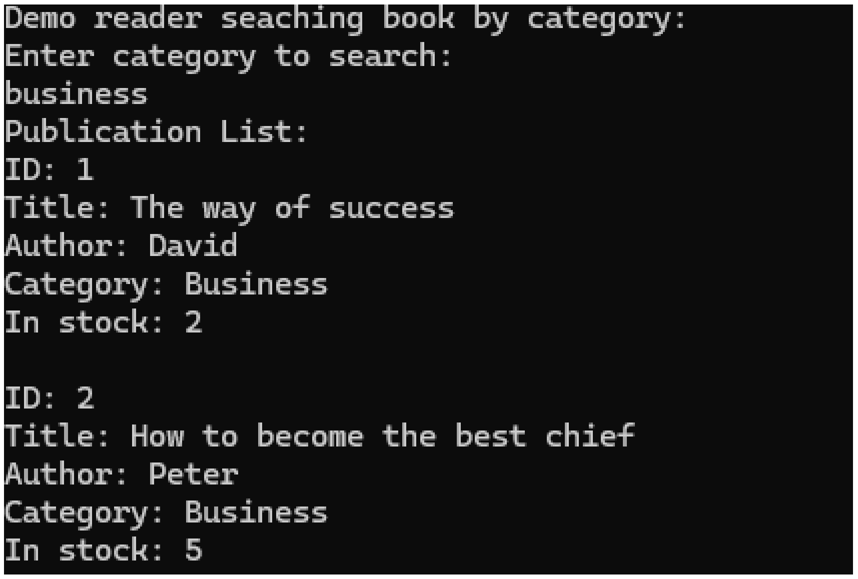
```
ID: 3
Title: 7 secrets of success
Author: John
Category: Inspiration
In stock: 4
```

Reader can search book by title:



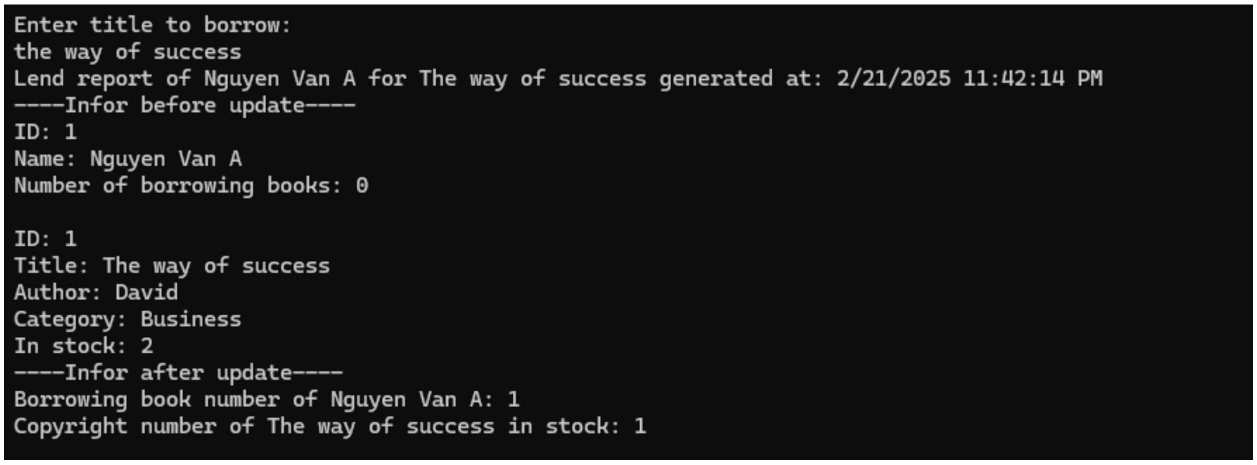
```
Demo reader seaching book by title:
Enter title to search:
the way of success
ID: 1
Title: The way of success
Author: David
Category: Business
In stock: 2
Demo reader seaching book by category:
```

Reader can search book by category:



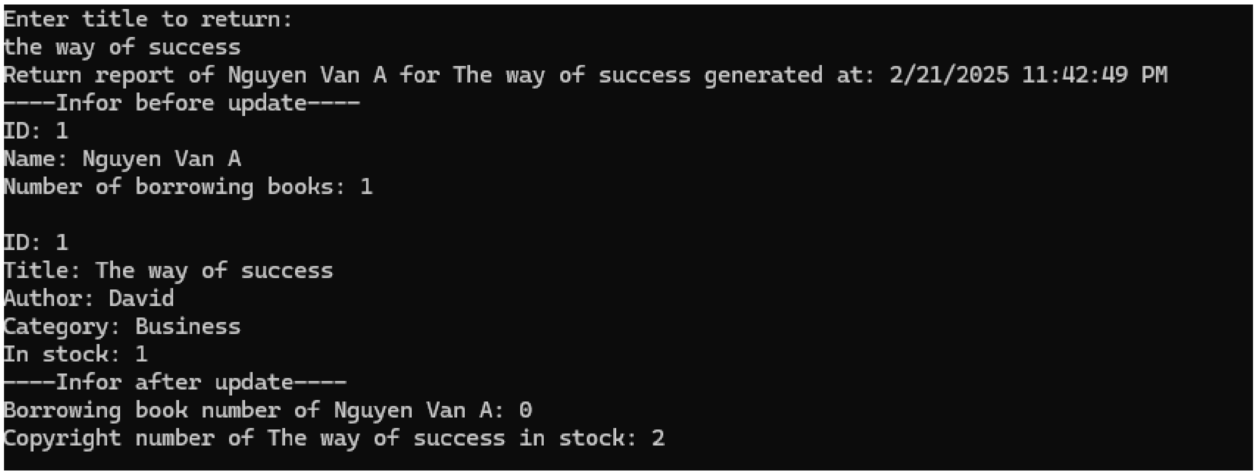
```
Demo reader seaching book by category:
Enter category to search:
business
Publication List:
ID: 1
Title: The way of success
Author: David
Category: Business
In stock: 2

ID: 2
Title: How to become the best chief
Author: Peter
Category: Business
In stock: 5
```

Reader can borrow book and the report will be generated:



```
Enter title to borrow:
the way of success
Lend report of Nguyen Van A for The way of success generated at: 2/21/2025 11:42:14 PM
----Infor before update----
ID: 1
Name: Nguyen Van A
Number of borrowing books: 0

ID: 1
Title: The way of success
Author: David
Category: Business
In stock: 2
----Infor after update----
Borrowing book number of Nguyen Van A: 1
Copyright number of The way of success in stock: 1
```

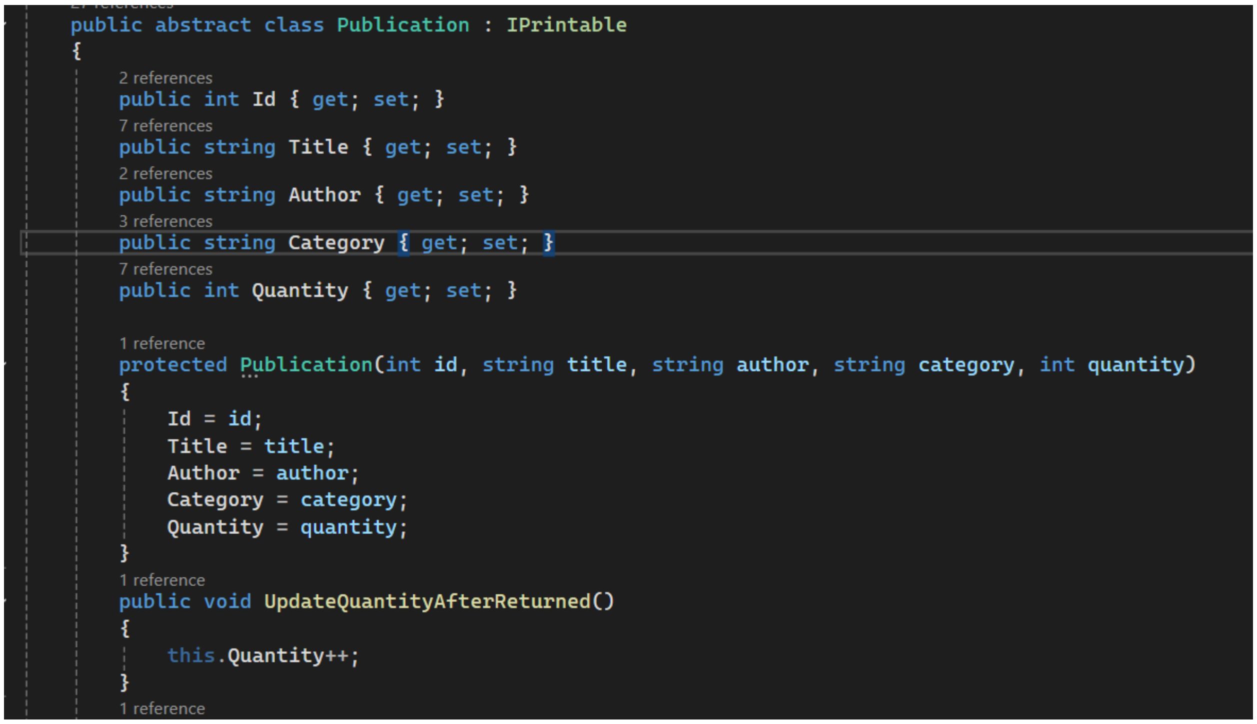
Reader can return book and the report will be generated:



```
Enter title to return:
the way of success
Return report of Nguyen Van A for The way of success generated at: 2/21/2025 11:42:49 PM
----Infor before update----
ID: 1
Name: Nguyen Van A
Number of borrowing books: 1

ID: 1
Title: The way of success
Author: David
Category: Business
In stock: 1
----Infor after update----
Borrowing book number of Nguyen Van A: 0
Copyright number of The way of success in stock: 2
```

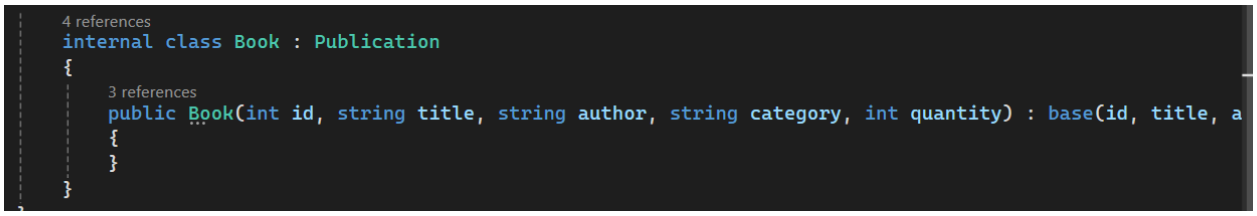
IV. Propose ways to extend the system.

By creating abstract publication class:



```
public abstract class Publication : IPrintable
{
    2 references
    public int Id { get; set; }
    7 references
    public string Title { get; set; }
    2 references
    public string Author { get; set; }
    3 references
    public string Category { get; set; }
    7 references
    public int Quantity { get; set; }

    1 reference
    protected Publication(int id, string title, string author, string category, int quantity)
    {
        Id = id;
        Title = title;
        Author = author;
        Category = category;
        Quantity = quantity;
    }
    1 reference
    public void UpdateQuantityAfterReturned()
    {
        this.Quantity++;
    }
    1 reference
```

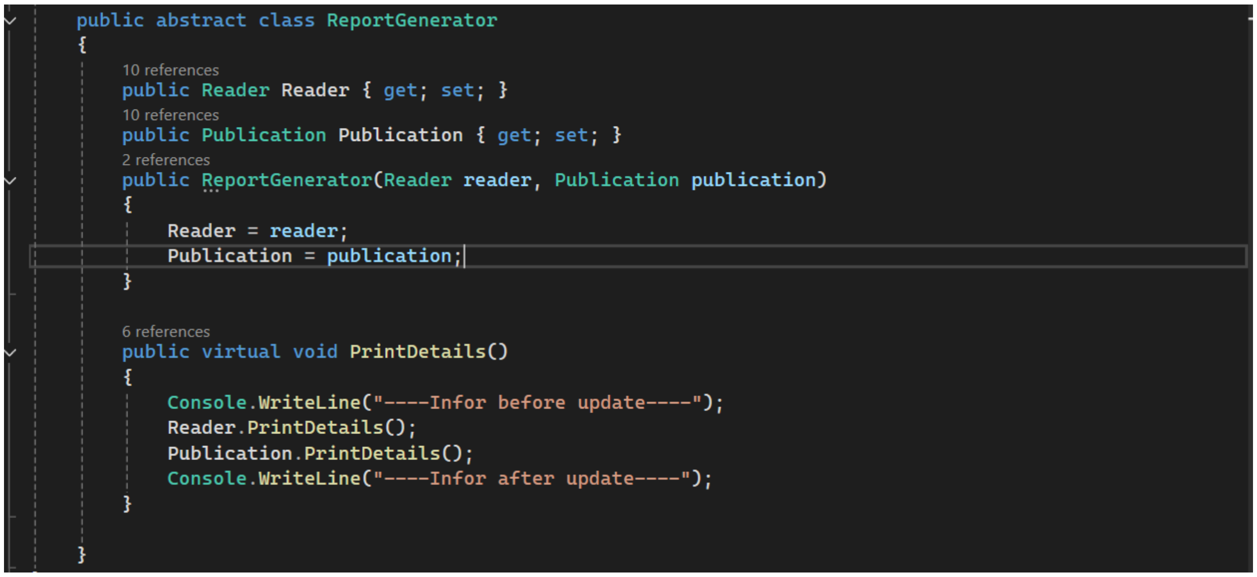
We can create different types of books in the future without any problems such as Book, eBook, Magazine, Newspaper,…:



```
4 references
internal class Book : Publication
{
    3 references
    public Book(int id, string title, string author, string category, int quantity) : base(id, title, a
    {
    }
}
```

We also can apply this system to manage many libraries later.

We can generate different types of reports when needed because we can extends the abstract class ReportGenerator:



```
public abstract class ReportGenerator
{
    10 references
    public Reader Reader { get; set; }
    10 references
    public Publication Publication { get; set; }
    2 references
    public ReportGenerator(Reader reader, Publication publication)
    {
        Reader = reader;
        Publication = publication;
    }

    6 references
    public virtual void PrintDetails()
    {
        Console.WriteLine("----Infor before update----");
        Reader.PrintDetails();
        Publication.PrintDetails();
        Console.WriteLine("----Infor after update----");
    }

}
```