International Journal of Advance Research in Science and Engineering
Vol. No.6, Issue No. 01 , January 2017
www.ijarse.com

IJARSE
ISSN (O) 2319 - 8354
ISSN (P) 2319 - 8346

CONDUCTING AND ANALYZING THE DESIGN PATTERNS OF LIBRARY MANAGEMENT SYSTEM WITH ITS FACTORY DESIGN

Teoh Wan Qi

Student, BSc (Hons) in Software Engineering, Asia Pacific University, Kuala Lumpur, Malaysia

ABSTRACT

Here, the Factory pattern and the Singleton pattern is implemented in this assignment after conducting analysis on different design patterns. Factory pattern is chosen because it is the most commonly used in the Java programming and it is more easily to be understand by the developer; whereas Singleton is use because it is the simplest pattern in the overall types of design pattern. Moreover, these patterns are also the solution to solve the issue found from the class diagram.

Keywords: Analysis, Design Pattern, Implementation, Java, Singleton

I. INTRODUCTION

The library management system involves different function such as managing book transactions and creating user. It will also require search function to enable the user to find their desire resource in the library. Hence, the design pattern of the system is important, as it not only help in managing the arrangement of code, but it also helps in managing the memory allocation of the code. Different design pattern can be used in different cases, where not all the design patterns are suitable for the library management system. In the library management system, it is found that the overall class diagram shown in “Fig.1” is quite complex and inflexible as some of the methods used in it can be modified by an implementation of design patterns. In this assignment, we will be implementing 2 design patterns into the system so to modify the current class diagram to a better approach.

1.1 Existing System Issues

Instance in the user class need to be access globally while being encapsulated because it is needed to be access frequently by the user class in order to check whether the user already exist in the system. Multiple object needed to be create, as there is different type of books with the same property but different attributes, such as Magazine and Journal. The code is coupled therefore it is not flexible if any changes needed to be made in the Book class

II. IDENTIFYING PATTERN

Identifying pattern is the process for solving the real problems based on factory pattern and singleton pattern.
The modified class diagram shown in “Fig.2”.

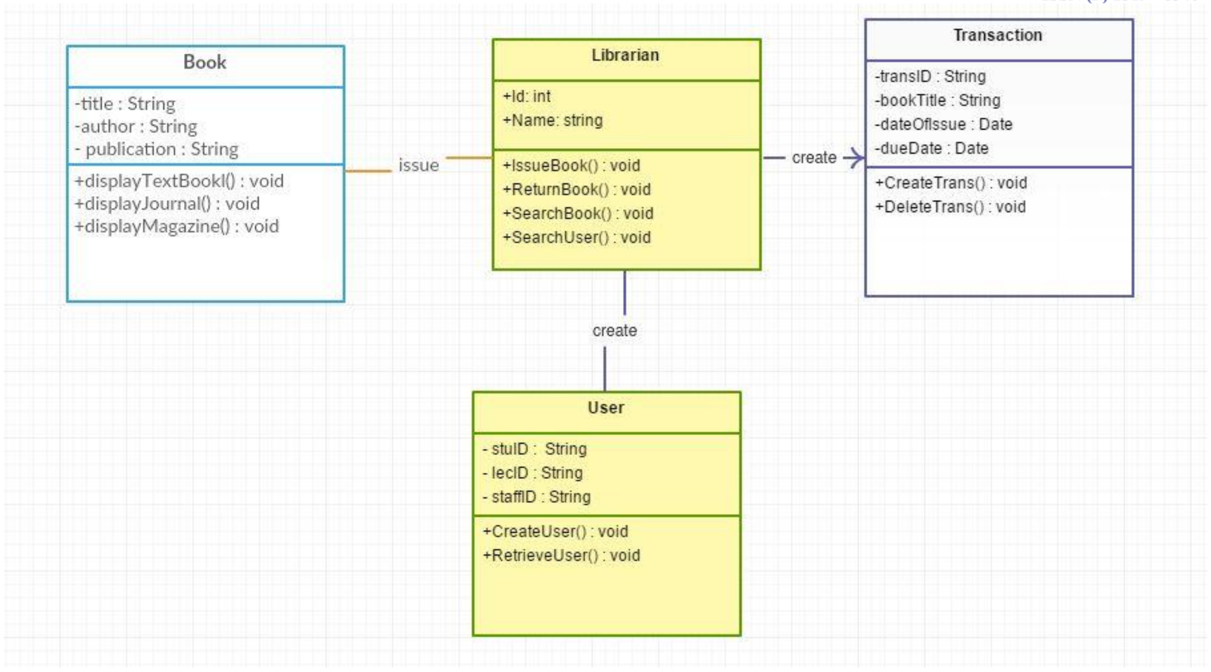


Figure 1. Class diagram

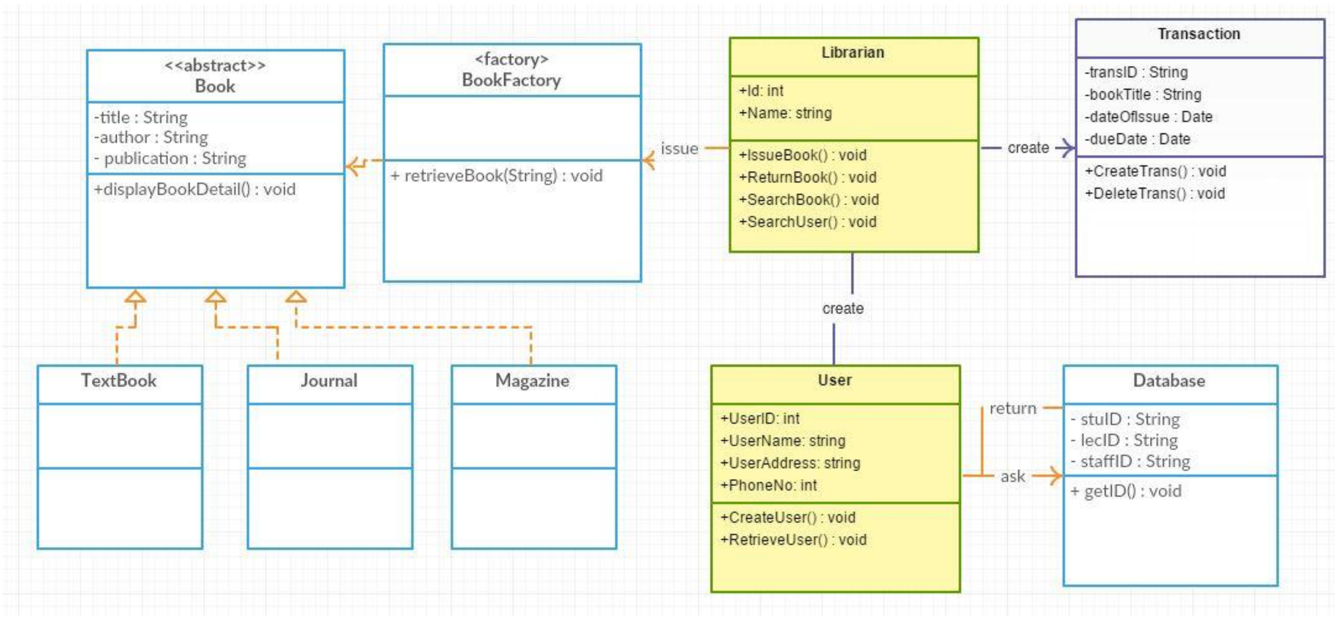


Figure.2 Modified Class diagram

III. SOLUTIONS WITH REFINED CLASS DIAGRAM

First, the book class is modified to become an abstract class and a set and get method is written in the class for every instance declared. A function named displayBookDetail() is declared in the book class as well for the factory class to access. Then, other subclasses such as magazine, journal and textbook class is inheriting to it. Next, a factory class named “BookFactory” is created and a method named retrieveBook with a String parameter is declared. Therefore, whenever the librarian class need to obtain the information from the book class, it needs to request from the factory class first. The “Fig.3” shown refined class digram.

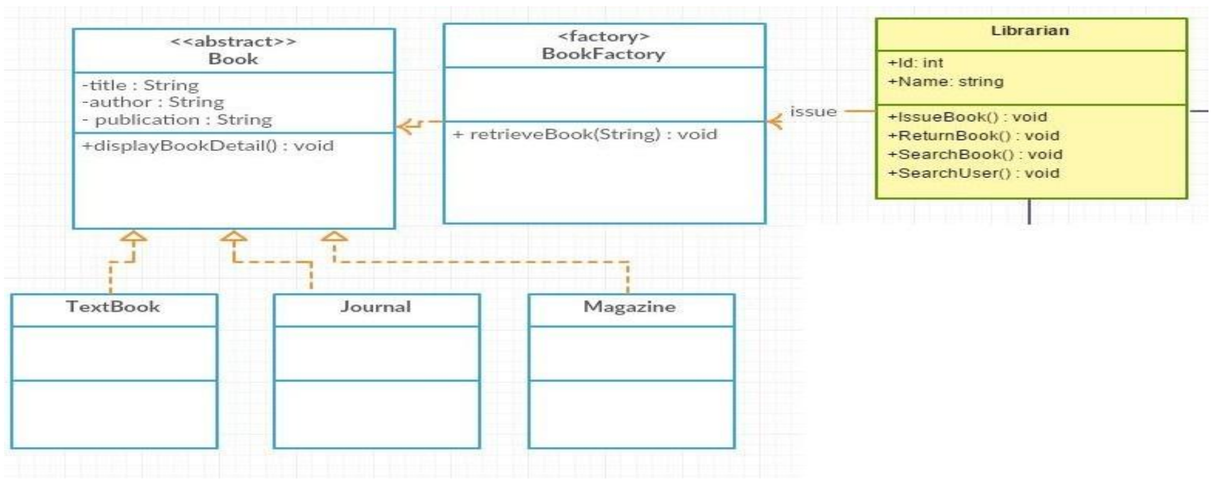


Figure.3 Refined class diagram

3.1 Justification

The factory pattern one of the pattern that is frequently used in Java programming. It is categorize as creational pattern as the pattern involving in creating objects. The factory pattern creates object without exposing the creation logic to the user and it will also refer to the new created object using interface [1]. The factory pattern is used in the book class because multiple classes such as magazine, journal and textbook need to be created and also to prevent the code for calling the method in the subclasses to be written everywhere in the different classes repeatly [2].

IV. SINGLETON PATTERN

4.1 Solution

First, a class named “Database” is created with a getID() and a private constructor. When the class was first accessed, an object will be created and stored inside its identifier. Hence, if the object is frequently accessed, new object will not be created, while the identifier will return the object that is created during the first invocation.

4.2 Refined Class Diagram

The following refined class diagram shown the user attributes and database shown in “Fig.4”.

IJARSE
ISSN (O) 2319 - 8354
ISSN (P) 2319 - 8346

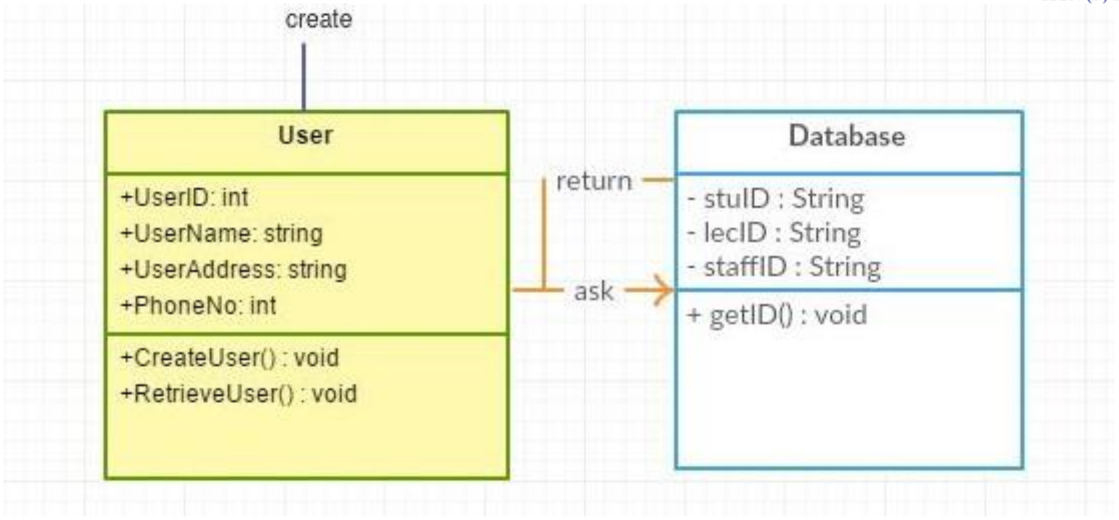
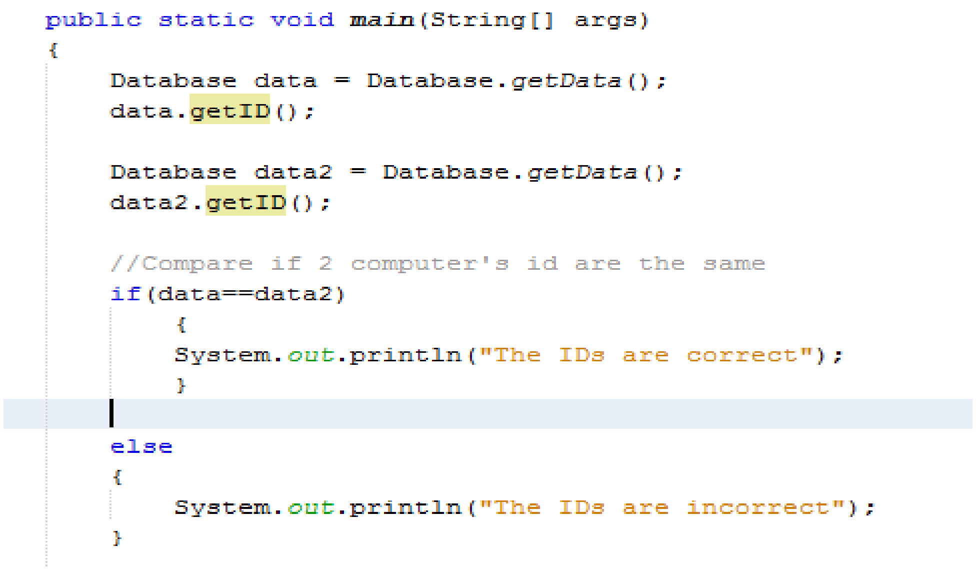


Figure.4 Refined Class diagram

4.3 Justification

In Java, Singleton is considered as one of the easiest pattern to be implemented into the code. This is also a creational pattern as it also involves in creating object. This pattern is used when there is one instance that is needed to be easily access frequently. The pattern of the singleton involves in creating object where only one object is created and provide a way to access the object without instantiate the object of the class [2]. In this scenario, 3 instances with String type which are stuID, lecID and staffID is created to enable the User class to check whether these ID are present in the system database so that no duplicate information will be create in the database.

V. SINGLETON PATTERN IN MAIN CLASS



```
public static void main(String[] args)
{
    Database data = Database.getData();
    data.getID();

    Database data2 = Database.getData();
    data2.getID();

    //Compare if 2 computer's id are the same
    if(data==data2)
        {
        System.out.println("The IDs are correct");
        }

    else
    {
        System.out.println("The IDs are incorrect");
    }
```

IJARSE
ISSN (O) 2319 - 8354
ISSN (P) 2319 - 8346

5.1 Output for Singleton Pattern

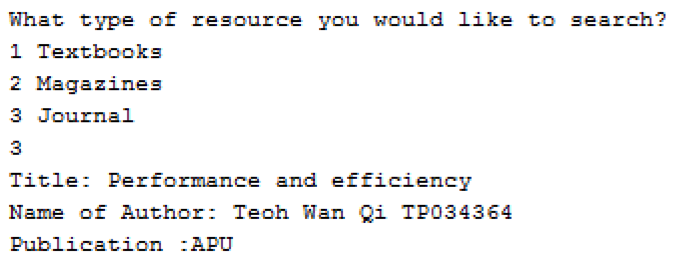
```
Student ID : 12345678b
Lecturer ID : 987654321a
Staff ID : 456789123c
Student ID : 12345678b
Lecturer ID : 987654321a
Staff ID : 456789123c
The IDs are correct
```

Comparison are made to show that whether or not the IDs exist in the system

VI. FACTORY PATTERN IN MAIN CLASS

```
    BookFactory bookfactory = new BookFactory();
    Book newbook = null;

    Scanner key = new Scanner(System.in);

    System.out.println("What type of resource you would like to search? ");
    System.out.println("1 Textbooks");
    System.out.println("2 Magazines");
    System.out.println("3 Journal");

    if(key.hasNextLine())
    {
        String bookType = key.nextLine();

        newbook = bookfactory.retriveBook(bookType);

    }

    if (newbook != null)
    {
        displayDetails(newbook);
    }
}

public static void displayDetails(Book aBook)
{
    aBook.displayBookDetails();
}
```

6.1 Output for the Factory pattern



```
What type of resource you would like to search?
1 Textbooks
2 Magazines
3 Journal
3
Title: Performance and efficiency
Name of Author: Teoh Wan Qi TP034364
Publication :APU
```

System enable user input the type of the book that he/she wants and the system will display all the information of that particular book type.

VII. ACKNOWLEDGMENT

Author would like to acknowledge Mr Umapathy Eaganathan, Faculty in Computing, Asia Pacific University, Malaysia for his constant support and encouragement to contribute in this International conference also for the publishing.

VIII. CONCLUSION

This paper explained and created a design pattern for library management system with different solutions provided with factory design and also provided with java codes.

REFERENCES

[1] www.tutorialspoint.com. (2016). *Design Pattern Factory Pattern*. [online] Available at: https://www.tutorialspoint.com/design_pattern/factory_pattern.htm [Accessed 25 Sep. 2016].

[2] Programmers.stackexchange.com. (2016). *Why should I use a factory class instead of direct object construction?*. [online] Available at: http://programmers.stackexchange.com/questions/253254/why-should-i-use-a-factory-class-instead-of-direct-object-construction [Accessed 26 Sep. 2016].