<Assignment 2>

Analysis and Design Document

Student: Topan Bianca - Corina

**Group: 32711**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 3

4. UML Sequence Diagrams 3

5. Class Design 3

6. Data Model 3

7. System Testing 3

8. Bibliography 3

1. Requirements Analysis

# Assignment Specification

**1. Objective**

The objective of this assignment is to allow students to become familiar with the Model View Controller architectural pattern and the Factory Method design pattern.

**2. Application Description**

Use Java/C# API to design and implement an application for the employees of a book store. The application should have two types of users (a regular user represented by the book store employee and an administrator user) which have to provide a username and a password in order to use the application.

The regular user can perform the following operations:

- Sell books.

The administrator can perform the following operations:

- CRUD on books (book information: title, author, genre, quantity, and price).

- **Obs.** Aici poate ramane doar ‘View Books’ (deci un tabel cu toate cartile)

**3. Application Constraints**



• The information about users and books can be optionally stored in files. Use the Model View Controller in designing the application.

• (The information about users, books and selling will be stored in multiple XML files. Use the Model View Controller in designing the application. Use the Factory Method design pattern for generating the reports.)

• (All the inputs of the application will be validated against invalid data before submitting the data and saving it.)

**4. Requirements**

- Create the analysis and design document (see the template).

- Implement and test the application.

**5. Deliverables**

- Analysis and design document.

- Implementation source files.

* 1. - (Readme file that describes the installation process of the application and how to use it: o how to install your application on a clean computer
  2. o how to access your application and with what users
  3. o images with all use cases and their scenarios implemented )

# Functional Requirements

The book store application have two types of users: **the book store employee** and **the administrator**. Both users can login at the application and perform the following operations:

* **the book store employee**: view account, sell books;
* **the administrator**: (CRUD)VIEW on books.

# Non-functional Requirements

Non-functional requirements are often called "[quality attributes](https://en.wikipedia.org/wiki/List_of_system_quality_attributes)" and can be divided into two categories:

* Execution qualities, such as safety, security and usability, which are observable during operation (at run time).
* Evolution qualities, such as [testability](https://en.wikipedia.org/wiki/Software_testability), extensibility, which are embodied in the static structure of the system. [1]

The application :

* must be easy to use,
* have a good response time,
* must provide security.

2. Use-Case Model

*Use-Case description format:*

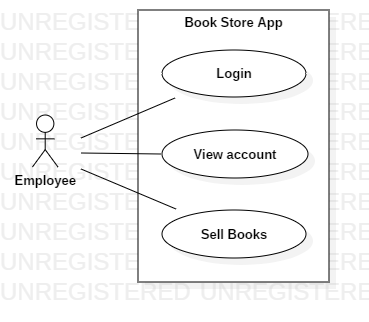
*Use case: Employee*

*Level: user-goal level*

*Primary actor: Employee*

*Main success scenario: <the steps of the main success scenario from trigger to goal delivery>*

*Extensions: <alternate scenarios of success or failure>*



*Use-Case description format:*

*Use case: Administrator*

*Level: user-goal level*

*Primary actor: Administrator*

*Main success scenario: <the steps of the main success scenario from trigger to goal delivery>*

*Extensions: <alternate scenarios of success or failure>*



3. System Architectural Design

**3.1 Architectural Pattern Description**

The architectural model used is Model View Controller.

**Model–View–Controller** (MVC) is an [architectural pattern](https://en.wikipedia.org/wiki/Architectural_pattern) commonly used for developing [user interfaces](https://en.wikipedia.org/wiki/User_interface) that divides an application into three interconnected parts. This is done to separate internal representations of information from the ways information is presented to and accepted from the user.[2] The MVC design pattern decouples these major components allowing for [code reuse](https://en.wikipedia.org/wiki/Code_reuse) and parallel development.

**Model**

The central component of the pattern. It is the application's dynamic data structure, independent of the user interface.[3] It directly manages the data, logic and rules of the application.

**View**

Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.

**Controller**

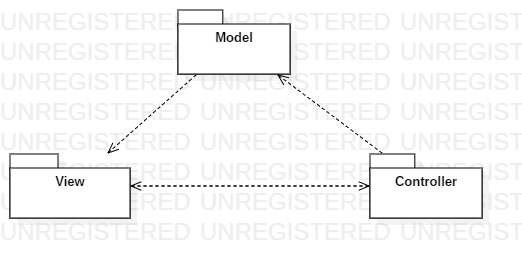
Accepts input and converts it to commands for the model or view.[4]

In addition to dividing the application into these components, the model–view–controller design defines the interactions between them.[5]

* The model is responsible for managing the data of the application. It receives user input from the controller.
* The view means presentation of the model in a particular format.
* The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

Traditionally used for desktop [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface) (GUIs), this architecture has become popular for designing [web applications](https://en.wikipedia.org/wiki/Web_application).[6] Popular programming languages like [JavaScript](https://en.wikipedia.org/wiki/JavaScript), [Python](https://en.wikipedia.org/wiki/Python_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [PHP](https://en.wikipedia.org/wiki/PHP), [Java](https://en.wikipedia.org/wiki/Java_(programming_language)), and [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)) have MVC frameworks that are used in web application development straight [out of the box](https://en.wikipedia.org/wiki/Out_of_the_box_(feature)).

**3.2 Diagrams**



4. UML Sequence Diagrams

**

5. Class Design

**5.1 Design Patterns Description**

In [software engineering](https://en.wikipedia.org/wiki/Software_engineering), **behavioral design patterns** are [design patterns](https://en.wikipedia.org/wiki/Design_pattern_(computer_science)) that identify common communication patterns among objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.[7]

Of this type of design I used: observer pattern and observable pattern.

**Observer and observable pattern**

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically.[8]

It specifies communication between objects: **observable**and**observers.**

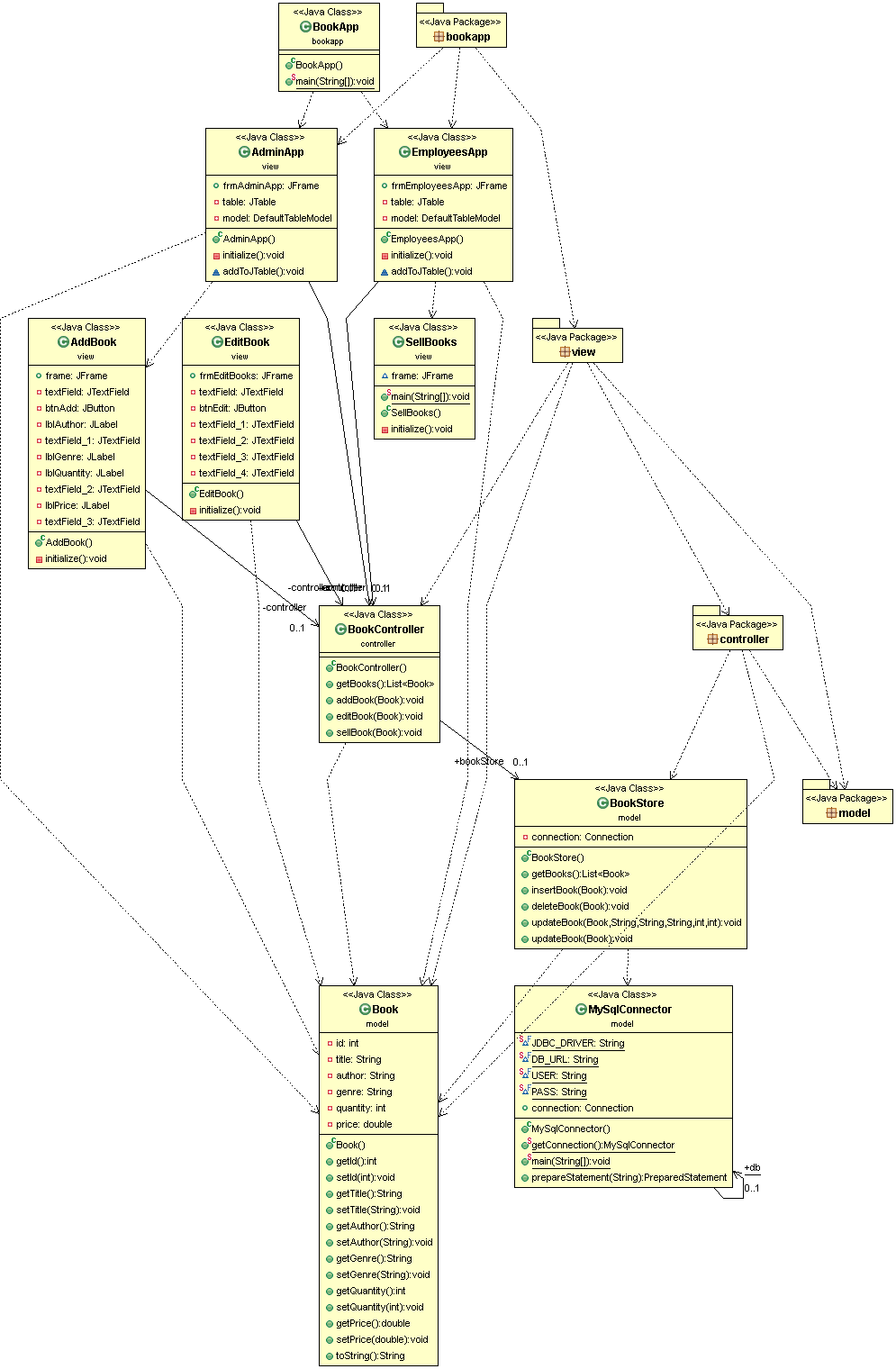
An **observable** is an object which notifies observers about the changes in its state.

1) [Observable](https://docs.oracle.com/javase/8/docs/api/java/util/Observable.html) is a class and [Observer](https://docs.oracle.com/javase/8/docs/api/java/util/Observer.html) is an interface.

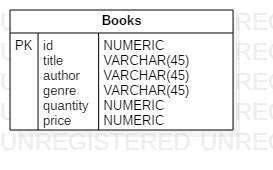
2) The Observable class maintains a list of Observers.

3) When an Observable object is updated, it invokes the update() method of each of its Observers to notify that, it is changed.

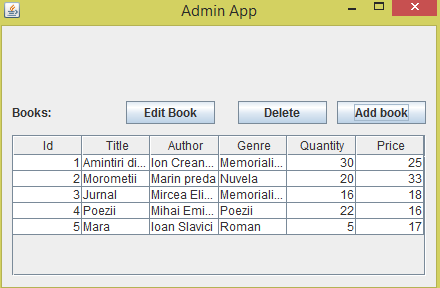
**5.2 UML Class Diagram**

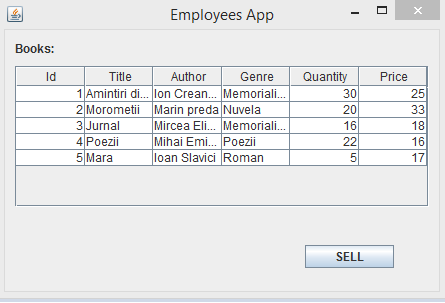
****

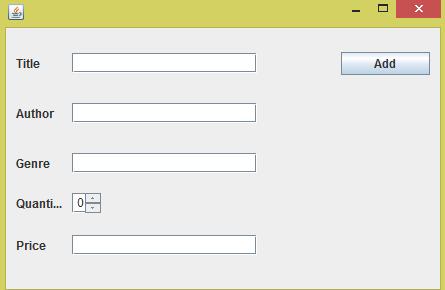
6. Data Model



7. System Testing







8. Bibliography

[1] <https://en.wikipedia.org/wiki/Non-functional_requirement>

[2] [The DCI Architecture: A New Vision of Object-Oriented Programming](http://www.artima.com/articles/dci_vision.html) – [Trygve Reenskaug](https://en.wikipedia.org/wiki/Trygve_Reenskaug) and [James Coplien](https://en.wikipedia.org/wiki/James_Coplien) – March 20, 2009.

[3]  Burbeck, Steve (1992) [Applications Programming in Smalltalk-80:How to use Model–View–Controller (MVC)](https://web.archive.org/web/20120729161926/http:/st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html)

[4] [Simple Example of MVC (Model–View–Controller) Design Pattern for Abstraction](http://www.codeproject.com/Articles/25057/Simple-Example-of-MVC-Model-View-Controller-Design)

[5] Buschmann, Frank (1996) Pattern-Oriented Software Architecture

[6] Davis, Ian. ["What Are The Benefits of MVC?"](http://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/). Internet Alchemy. Retrieved 2016-11-29.

[7] <https://en.wikipedia.org/wiki/Behavioral_pattern>

[8] <https://www.tutorialspoint.com/design_pattern/observer_pattern.htm>