National Theater of Cluj

Student: Cârcu Bogdan

**Group: 30431**

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

Use JAVA/C# API to design and implement an application for the National Theater of Cluj. The application should have two types of users (a cashier user represented and an administrator) which must provide a username and a password to use the application.

# Functional Requirements

The administrator user can perform the following operations:

CRUD on cashiers’ information.

CRUD on the list of shows that are performed at the theater. Keep track of the Genre (Opera, Ballet), Title, Distribution list (a long string is enough), Date of the show and the Number of tickets per show.

From time to time he can export all the tickets that were sold for a certain show (either in a csv or xml file).

The cashier can perform the following operations:

Sell tickets to a show. A ticket should hold information about the seat row and seat number.

The system should notify the cashier that the number of tickets per show was not exceeded.

A cashier can see all the tickets that were sold for a show, cancel a reservation or edit the seat.

# Non-functional Requirements

The data will be stored in a database.

Use the Layers architectural pattern to organize the application.

Passwords are encrypted when stored to the database with a one-way encryption algorithm.

Use **factory method** for export to csv/xml.

2. Use-Case Model

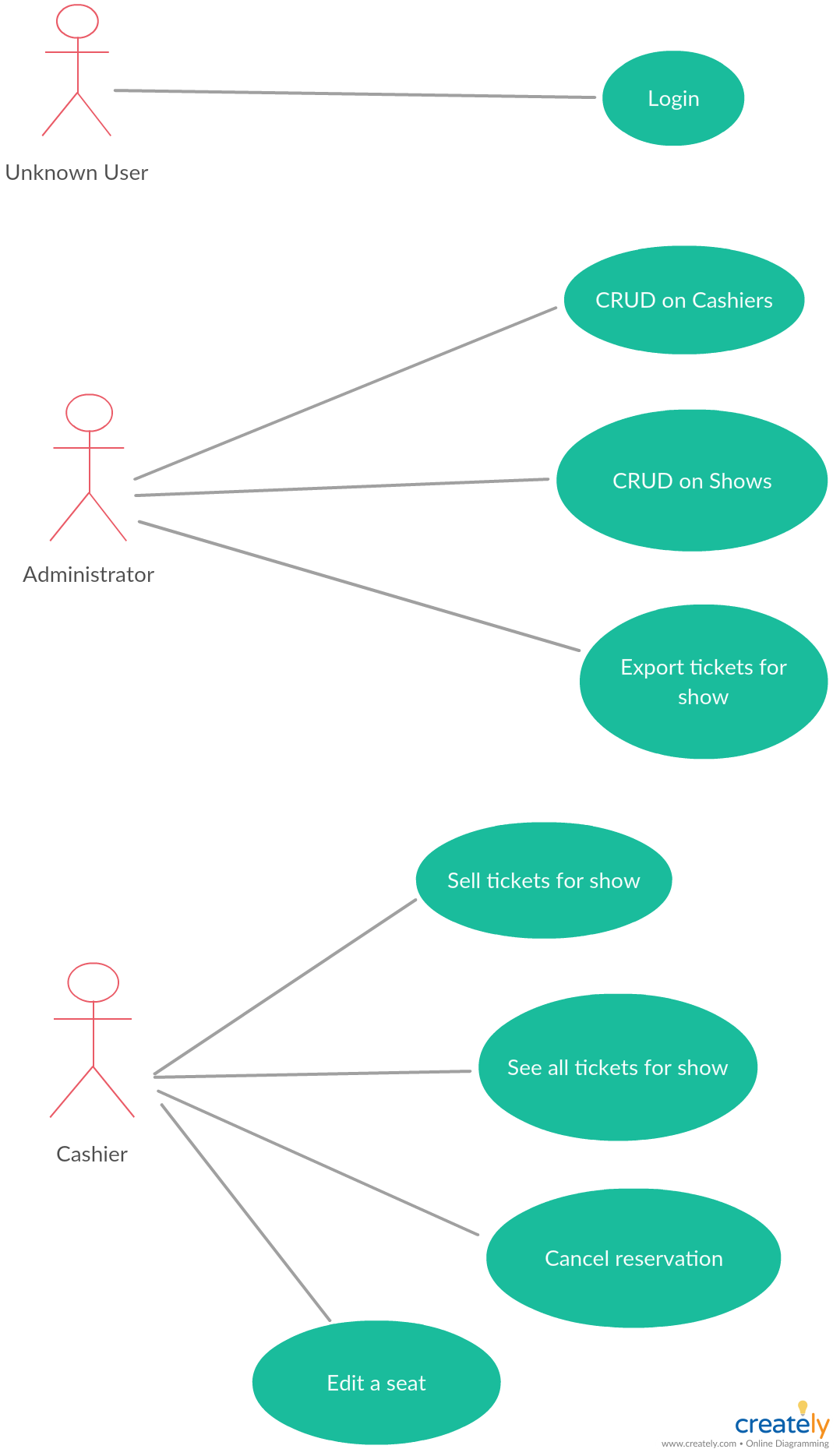
Use case: System usage

Level: user-goal level

Primary actors: Unknown User, Administrator, Cashier

Main success scenario: all actions are performed successfully

Extensions: error messaging in case of failure, system responses for tickets’ validation



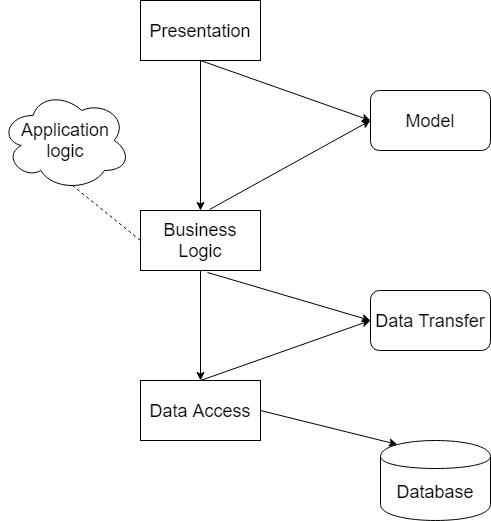
3. System Architectural Design

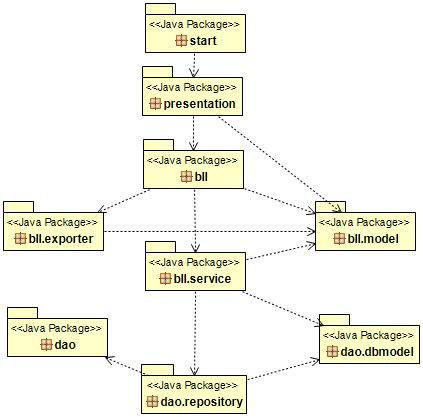
**3.1 Architectural Pattern Description**

**Layered Architecture**

* This architecture is a client-server architecture in which presentation, application processing, and data management functions are separated.
* Every layer can use the functionalities of the layer below but not vice-versa

**3.2 Diagrams**

****

**

Database: MySQL

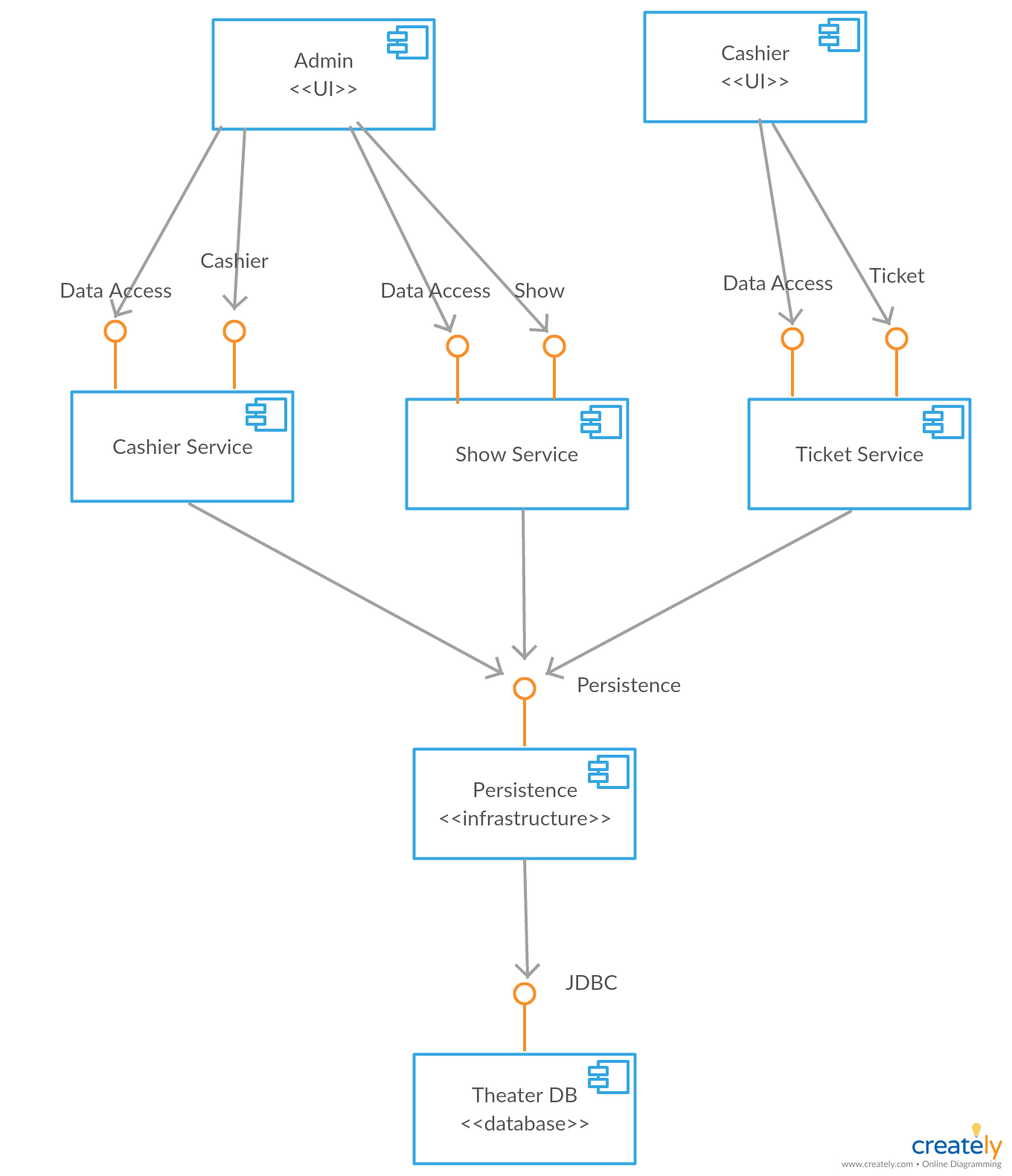
Data access: **dao** package

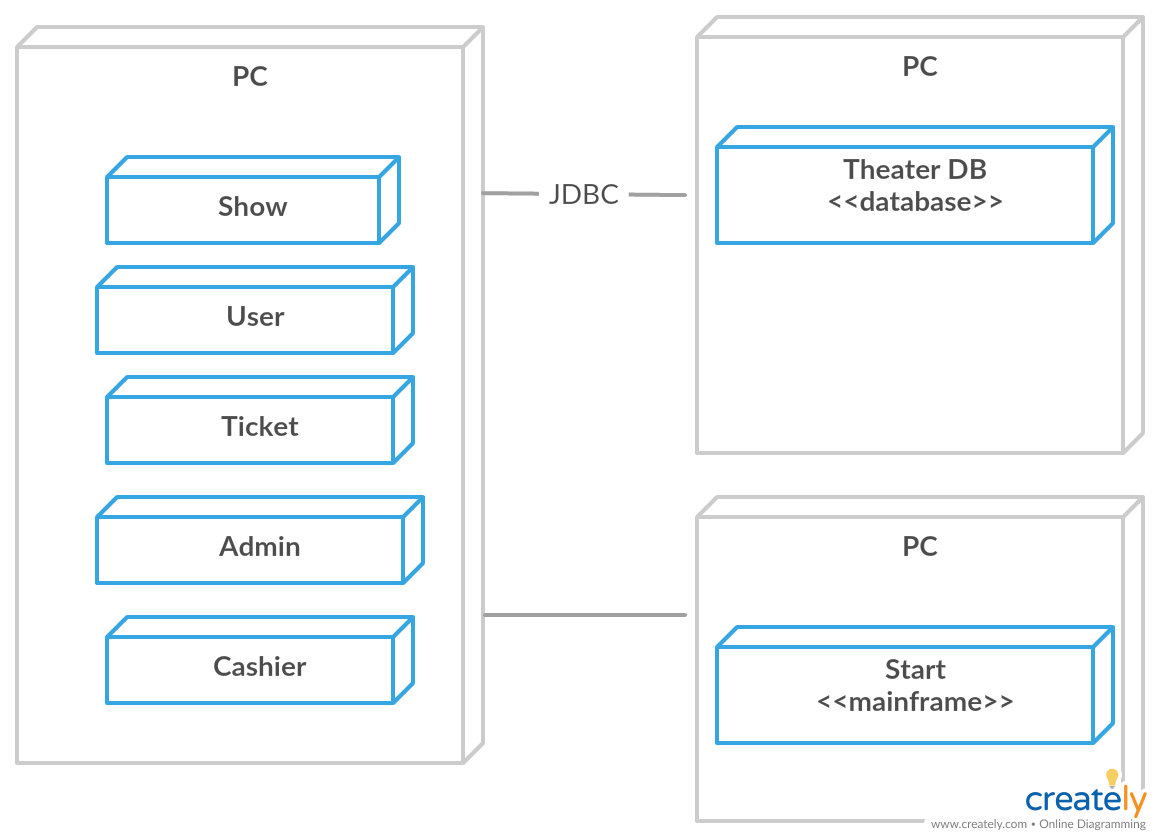
Data transfer: **dao.dbmodel** sub-package

Business Logic + Application logic: **bll** package

Model: **bll.model** sub-package

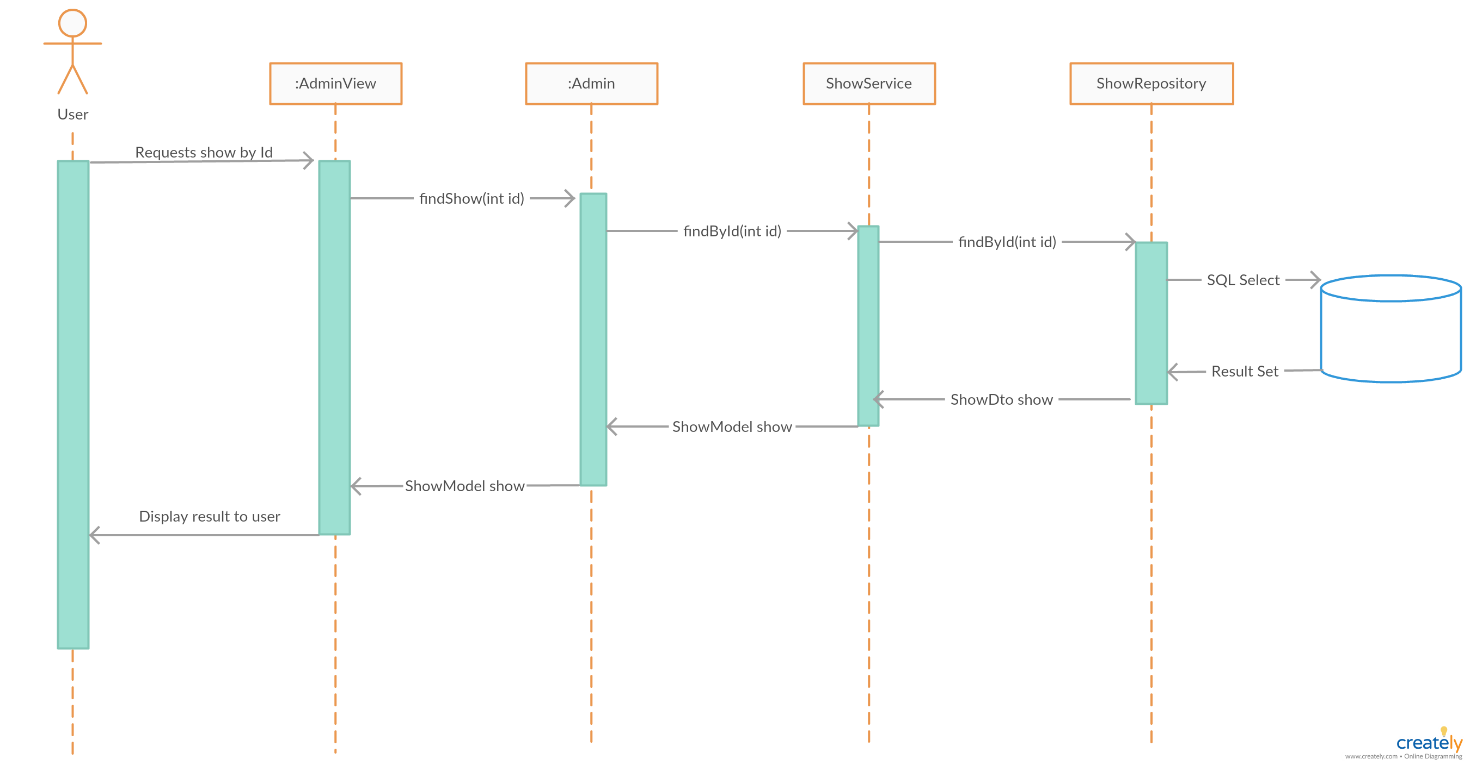
Presentation: **presentation** package





4. UML Sequence Diagrams

* Admin selects a show by its id



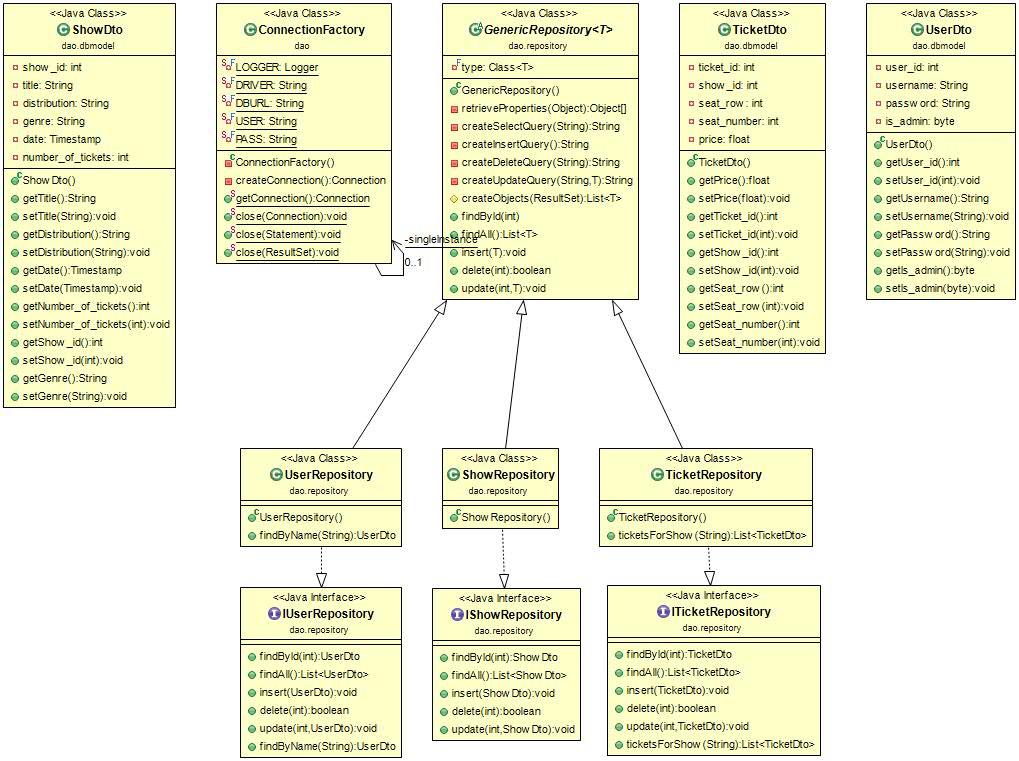
5. Class Design

**5.1 Design Patterns Description**

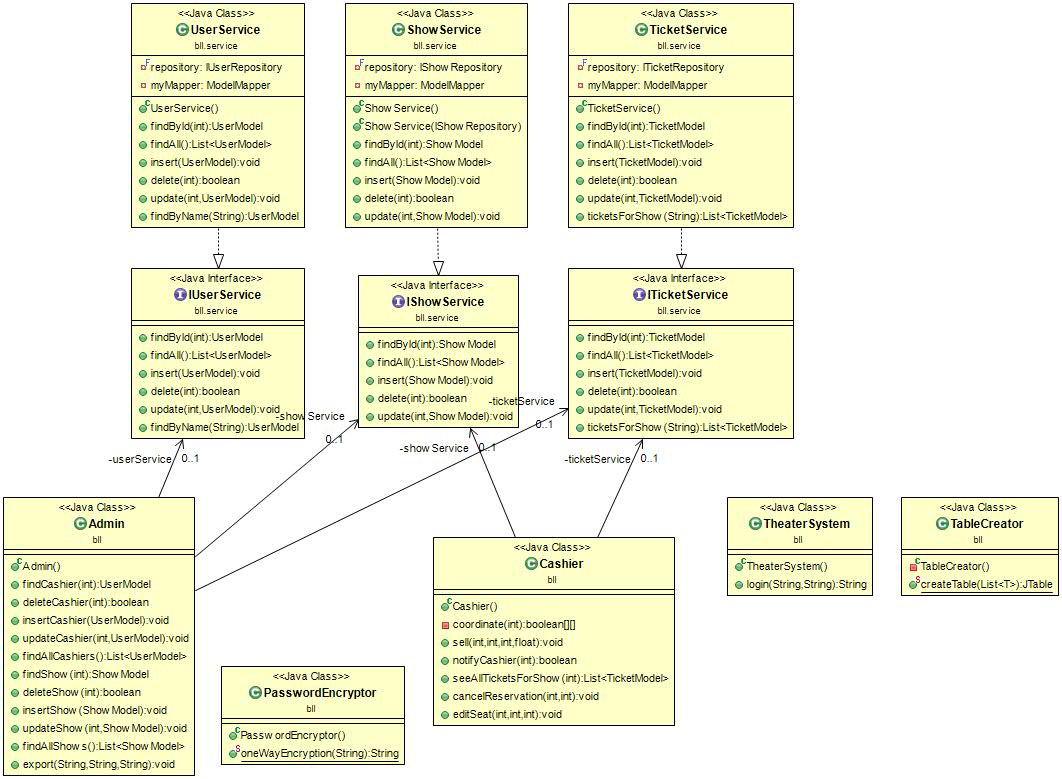
* In [class-based programming](https://en.wikipedia.org/wiki/Class-based_programming), the **Factory Method Pattern** is a [creational pattern](https://en.wikipedia.org/wiki/Creational_pattern) that uses factory methods to deal with the problem of [creating objects](https://en.wikipedia.org/wiki/Object_creation) without having to specify the exact [class](https://en.wikipedia.org/wiki/Class_(computer_programming)) of the object that will be created. This is done by creating objects by calling a factory method—either specified in an interface and implemented by child classes, or implemented in a base class and optionally overridden by derived classes—rather than by calling a [constructor](https://en.wikipedia.org/wiki/Constructor_(object-oriented_programming)).
* The **Singleton** **Pattern** involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.

**5.2 UML Class Diagram**

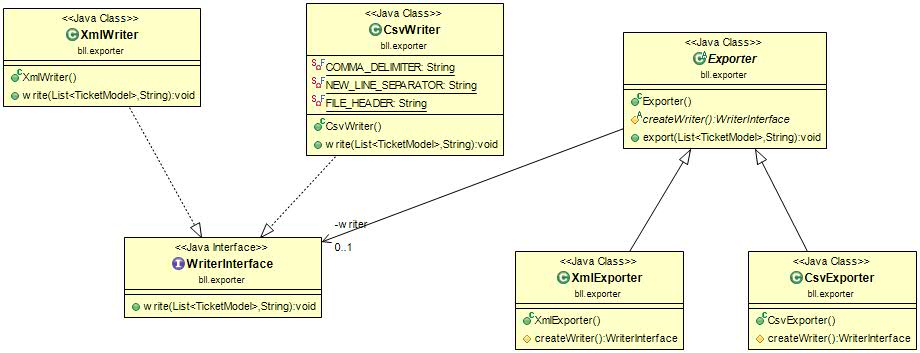
DAO – “ConnectionFactory” is an exemplification of **Singleton**

**

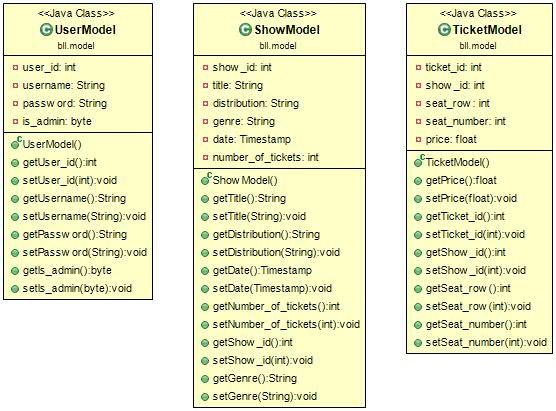
BLL

**

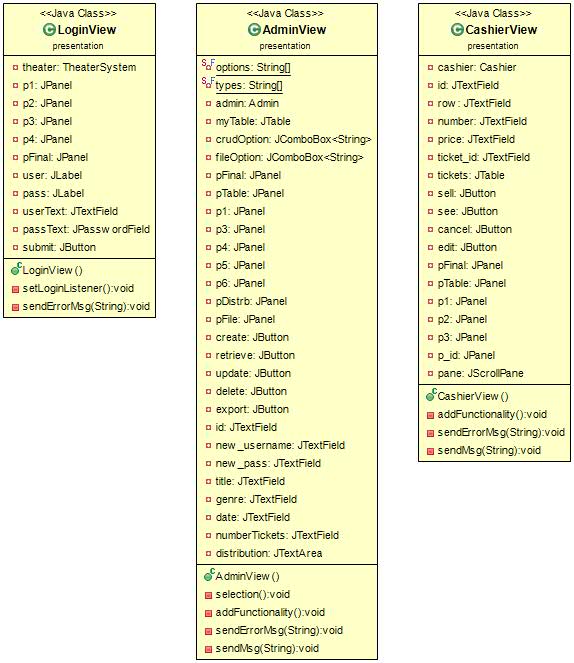
Exporter – exemplification of **factory method**

****

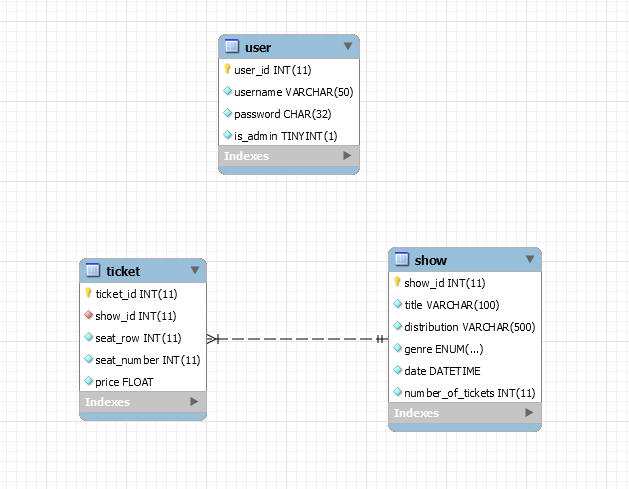
Model



Presentation

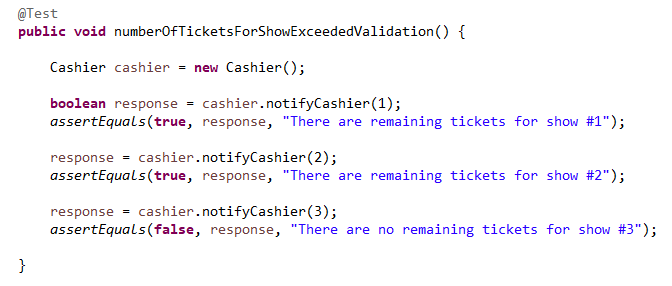


6. Data Model

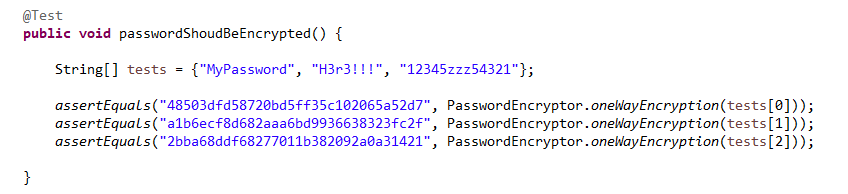
**

7. System Testing

* number of tickets for show exceeded validation



* the encryption algorithm.



8. Bibliography

* Layered Architecture

<https://dzone.com/articles/layered-architecture-is-good>

* Singleton

<https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm>

* Factory Method

<https://en.wikipedia.org/wiki/Factory_method_pattern>

* Mockito

<https://www.mkyong.com/unittest/unit-test-what-is-mocking-and-why/>