КУРСОВ ПРОЕКТ

ТЕМА: Система за управление на заведение (бар)

1. **Функционалност.**

Функционалността, която приложението предлага е следната:

1.1. Възможност за добавяне на потребители. Потребителите могат да бъдат в следните роли: сервитьор, барман и управител.

1.2. Да може да се добавят нови артикули (коктейли, напитки и т.н.), с съответните им детайли (име, цена и т.н.).

1.3. Сервитьорите могат да приемат поръчки – в една поръчка може да има различен на вид и брой артикули.

1.4. Барманите съответно трябва да могат да виждат заявените поръчките в системата и да ги „изпълняват”. Поръчките да имат статус (напр. „чакаща”, „приета”, „изпълнена” и „пресрочена”). Ако дадена поръчка не се поеме от някой барман в следващите 5минс, да се изведе някакво съобщение с напомняне. Ако съответно поръчката не се приготви за 5минс от барман-а, да се запише като пресрочена. Трябва да се подсигури, че 2ма бармани не могат да изпълнят едновременно една и съща поръчка.

1.5. Управителите да могат да извеждат определен тип справки за дейността с цел анализиране на бизнес дейността. Например: оборот (по ден/месец/часови интервал и т.н.), продажби по тип артикули, закъснели поръчки и т.н.

1. **Изисквания към имплементацията.**

2.1. Приложението да работи в J2EE среда

2.2. Да се ползва Java уеб графичен потребителски интерфейс

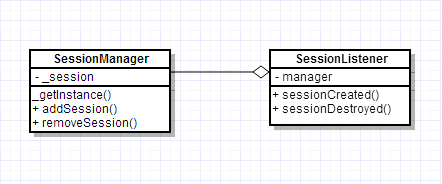
2.3. Потребителите трябва да могат да достъпват системата през логин панел.

2.4. Като хранилище за данните да се ползва база данни.

2.5. Да се приложат поне 5 шаблона за проектиране и да се документира връзките и отношенията между тях с диаграми на класовете.

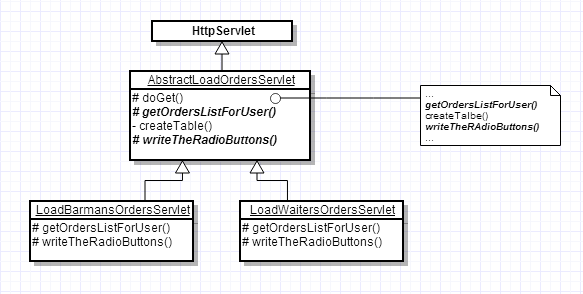
1. **Използвани шаблони в реазлизацията на системата:**
   1. Singleton
   2. Template Method
   3. Factory Method
   4. Observer
   5. Facade
2. **Документиране на връзките между класовете, реализирали шаблоните.**
   1. **Singleton:**

* *Singleton* (SessionManager) – съдържа лична (private) инстанция, която се достъпва чрез статичен публичен метод.
* *Client* (SessionListener) – класът има достъп само през статичния метод getInstance().



* 1. **Template Method:**
* *Abstract class* (AbstractLoadOrdersServlet)
  + - * 1. декларира абстрактни методи (getOrdersListForUser() и writeTheRadioButtons()), които биват имплементирани в конкретните наследници;
        2. дефинира template метод (doGet()), описващ структурата на “алгоритъма”.
* *Concrete class* (LoadBarmansOrdersServlet, LoadWaitersOrdersServlet)

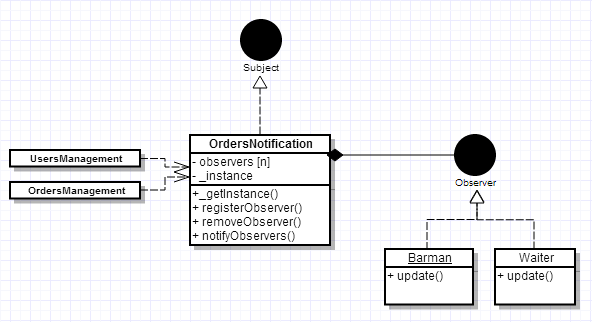
1. имплементира абстрактните методи, наследени от базовия клас, като всеки наследник предоставя конкретна за себе си реализация.



* 1. **Factory Method**
* *Creator* (UserFactory) – дефинира *параметризиран фабричен метод*, който според аргумента си определя какъв конкретен тип продукт да създаде.
* *Use case* (UserManagement) – предоставя операция, която извиква статичния метод на класа UserFactory

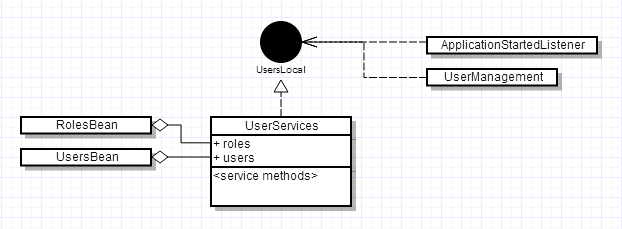


* 1. **Observer**
* *Subject –* интерфейс, който декларира необходимите методи (registerObserver(), removeObserver(), notifyObservers());
* *Concrete Subject* (OrdersNotification) *–* клас (в случая Singleton ), който имплементира методите, дефинирани в интерфейса Subject;
* *Observer –* интерфейс, предоставящ метода update().
* *Concrete Observer* (Barman, Waiter) *–* конкретен клас, който имплементира интерфейса Observer, осигуряващ специфична реализация на имплементирания метод.
* *Use case* (UsersManagement, OrdersManagement) – клиентски клас, който чрез извикване на статичния метод на Subjcet, съответно добавя/изтрива и осведомява наблюдателите.

****

* 1. **Facede**
* *Façade* (UserServices) – делегира клиентската заявка към правилния клас, който да извърши необходимата операция;
* *Subsystem class* (RolesBean, UsersBean)

1. имплементира „скрита” за другите системи функционалност;
2. изпълнява метод, поискан от Façade обекта;



Subsystem classes