**Ресторан(1)**

Давай напишем программу для автоматизации работы в ресторане.

Пусть задание сформулировано так — директор ресторана хочет, чтобы:  
1. на каждом столике лежал планшет, через который можно было бы сделать заказ;  
2. пока заказ готовится, на планшете показывалась реклама;  
3. в конце рабочего дня была возможность посмотреть различную статистику:  
а) загрузки повара;  
б) сумму выручки за заказы;  
в) сумму выручки за показы рекламы.

Мы будем разрабатывать ядро, без UI. UI ты потом сможешь дописать самостоятельно, сейчас ограничимся консольным интерфейсом.  
Также мы часть данных захардкодим, ты потом сможешь прикрутить БД, и работать с ней.

Выделим несколько фич, которые будем реализовывать:

Первая:  
— созданный посетителем заказ будет автоматически поступать к повару;  
— повар будет готовить его какое-то время и отмечать приготовленным;  
— после этого официант будет относить его.

Вторая:  
— подобрать нужные рекламные ролики из списка оплаченных;  
— отображение рекламных роликов во время приготовления заказа;  
— максимизировать прибыль от показа рекламы.

Третья:  
— подсчет статистики;  
— отображение статистики директору.

Начнем с первой фичи — автоматизация обработки заказа:

Итак..  
1. Создай класс Restaurant с методом main.

2. Начнем с планшета, создадим класс Tablet, в котором создадим поле final int number — это номер планшета, чтобы можно было однозначно установить, откуда поступил заказ. Номер планшета должен инициализироваться в конструкторе переданным параметром.

3. В классе Tablet создадим метод public void createOrder(), который будет создавать заказ из тех блюд, которые выберет пользователь.  
Для этого создадим класс Order, который поместим в пакет kitchen.

4. В классе Order (*заказ*) должна быть информация, относящаяся к списку выбранных пользователем блюд.  
Т.е. где-то должен быть список всех блюд, и должен быть список выбранных блюд в классе Order.  
В классе Order нужны поля private final Tablet tablet и protected List<Dish> dishes.  
Конструктор должен принимать один параметр типа Tablet и инициализировать поле tablet.

5. Нам нужен класс Dish(*Блюдо*), создадим его в пакете kitchen. Пусть это будет enum со списком блюд: Fish, Steak, Soup, Juice, Water.

6. Чтобы пользователь мог выбрать себе блюда, нужно их все ему отобразить. Для этого в Dish создай метод public static String allDishesToString(), который сформирует строку из всех блюд.

Пример: «*Fish, Steak, Soup, Juice, Water*«. Формируй строку динамически.

Требования:  
1. В классе Restaurant должен быть создан метод public static void main(String[] args).  
2. В классе Tablet должно быть создано поле final int number.  
3. Конструктор класса Tablet должен принимать 1 параметр типа int и инициализировать поле number.  
4. В пакете kitchen должен быть создан enum Dish с элементами Fish, Steak, Soup, Juice, Water.  
5. В Dish должен быть создан статический метод allDishesToString, динамически формирующий строку из всех блюд.  
6. В пакете kitchen должен быть создан класс Order, содержащий в себе список блюд(dishes), а также планшет(tablet).  
7. Конструктор класса Order должен принимать один параметр типа Tablet и инициализировать поле tablet.  
8. В классе Tablet должен существовать метод public void createOrder().

**Ресторан(2)**

1. Мы много работаем с консолью. Пора создать единую точку взаимодействия.  
Создай класс ConsoleHelper с единственным BufferedReader, через который будем работать с консолью.  
Запомни, этот класс не хранит никаких данных и состояний, поэтому все методы будут статическими.  
Создай в нем три метода:  
— writeMessage(String message) — для вывода message в консоль  
— String readString() — для чтения строки с консоли  
— List<Dish> getAllDishesForOrder() — просит пользователя выбрать блюдо и добавляет его в список.  
Выведи список всех блюд и попроси пользователя ввести строку — название блюда.  
Введенное ‘exit‘ означает завершение заказа.  
В случае, если введенное блюдо не представлено в меню, выведи сообщение о том, что такого блюда нет и продолжи формирование заказа.  
Исключения ввода-вывода бросай выше, на этом уровне не понятно, что с ними делать.

2. Сделай рефакторинг — работа с консолью должна быть только через класс ConsoleHelper.

3. Вернемся к классу Order: в нем есть ссылка на планшет, и еще есть список выбранных блюд.  
Инициализируй список dishes в конструкторе, вызвав метод getAllDishesForOrder из ConsoleHelper.

4. Перепиши метод toString в классе Order. Пусть он возвращает пустую строку, если нет блюд в заказе, иначе вывод должен быть аналогичным примеру в порядке добавления блюд. Используй ConsoleHelper.  
Также измени метод toString в классе Tablet (*внутри класса* Tablet *нажмите* Alt+Insert -> toString()).  
Пример:  
Your order: [Juice, Fish] of Tablet{number=5}

5. У нас все завязано на работу с консолью. Однако, при возникновении исключений, наше приложение умрет.  
Чтобы узнать причину — добавим в Tablet статическое поле logger типа *java.util.logging.Logger*, инициализированное именем класса (Logger.getLogger(Tablet.class.getName())).

6. В методе createOrder класса Tablet обработаем исключения ввода-вывода.  
Запишем в лог «*Console is unavailable.*«. Уровень лога — SEVERE — это самый серьезный уровень, мы не можем работать.  
Также в методе createOrder класса Tablet должен быть создан новый заказ.

7. Надо начинать тестировать наше приложение.  
Добавьте в main создание планшета и создание заказа — new Tablet(5).createOrder();

Требования:  
1. Метод writeMessage класса ConsoleHelper должен выводить полученную строку в консоль.  
2. Метод readString класса ConsoleHelper должен возвращать строку считанную с консоли.  
3. Метод getAllDishesForOrder класса ConsoleHelper должен возвращать список блюд выбранных пользователем.  
4. В конструкторе класса Order список dishes должен быть инициализирован результатом работы метода getAllDishesForOrder.  
5. Метод toString в классе Order должен возвращать содержимое заказа согласно условию задачи.  
6. Поле logger в классе Tablet должно быть инициализировано с помощью метода Logger.getLogger() с именем текущего класса в качестве параметра.  
7. В методе createOrder класса Tablet должен быть создан новый заказ, в качестве параметра передай текущий планшет.  
8. В случае возникновения IOException в процессе создания заказа - в лог должно быть записано сообщение "Console is unavailable.", уровень - SEVERE.  
9. В методе main класса Restaurant должен быть создан новый планшет и с его помощью создан новый заказ.  
10. Метод toString в классе Tablet должен возвращать информацию о планшете (смотри пример в задании).

### Ресторан(3)

Сейчас мы можем создавать заказы. Давай сделаем так, чтобы они автоматически поступали к повару.  
Есть много различных способов реализации данной функциональности.  
Почитай про паттерн **Observer** — ***http://en.wikipedia.org/wiki/Observer\_pattern***  
Он уже реализован в **java**, и мы его будем использовать.

1. Создадим класс **Cook**(Повар) в пакете kitchen, он будет готовить. Пусть в конструкторе приходит его имя, которое выводится методом toString.

2. Tablet создает заказы, а **Cook** их обрабатывает. Расставь правильно **Observer** и **Observable** между **Tablet** и **Cook**.

3. Метод void update(Observable observable, Object arg), который необходимо реализовать, принимает два параметра.  
— **observable** — объект, который отправил нам значение  
— **arg** — само значение, в нашем случае — это объект Order  
На данном этапе мы будем лишь имитировать обработку и выведем в консоль «***Start cooking —***» **+** order

3. Пишем main.  
Для объекта **Observable** добавляем свой объект **Observer**. См. **п.2** и описание паттерна в **wikipedia**  
Называем повара, имя не влияет на тесты. В моем варианте — это **Amigo** : )

Сверим выводы в консоль. **Пример моего вывода:**  
Your order: [Soup] of Tablet{number=5}  
Start cooking - Your order: [Soup] of Tablet{number=5}

4. Не забудь сразу после создания заказа и вывода информации о нем в консоль (найдите это место в коде) сделать следующее:  
4.1. Установить флаг setChanged()  
4.2. Отправить обсерверу заказ — notifyObservers(**order**);

5. Также внесем небольшое изменение. Сделай так чтобы метод createOrder возвращал текущий заказ или **null**, если заказ создать не удалось.

Требования:  
1. Конструктор класса Cook должен принимать в качестве параметра строку и инициализировать имя повара (поле name).  
2. Метод toString класса Cook должен возвращать значение поля name.  
3. Класс Cook должен реализовывать интерфейс Observer.  
4. Класс Tablet должен быть потомком класса Observable.  
5. В методе createOrder класса Tablet должен быть вызван метод setChanged.  
6. В методе createOrder класса Tablet должен быть вызван метод notifyObservers с текущим заказом в качестве параметра.  
7. Метод update в классе Cook должен выводить на экран строку форматированную согласно заданию.  
8. В методе main класса Restaurant должен быть создан новый повар и добавлен планшету в качестве наблюдателя с помощью метода addObserver.

### Ресторан(4)

Что мы имеем? Повар приготовил блюда, пора их уносить, а официант не знает об этом.

Нужно уведомить официанта, что пора забирать заказ с кухни.

Ситуация аналогична предыдущему заданию, поэтому снова будем использовать паттерн Observer.

1. Создадим класс Waiter(Официант) в пакете kitchen, он будет относить заказы назад к столику. Официант будет безымянным.

2. Cook(Повар) готовит заказы, а Waiter их обрабатывает. Расставь правильно Observer и Observable между Waiter и Cook.

3. Метод void update будет выводить в консоль сообщение о том, какой заказ и кем был приготовлен:

order + " was cooked by " + cook

4. В классе наследнике Observable перед отправкой заказа сделаем следующее:

4.1. Установим флаг setChanged()

4.2. Отправим наблюдателю заказ - notifyObservers(order);

5. Пишем main.

Для объекта Observable добавляем свой объект Observer. См. п.2 и описание паттерна в wikipedia

Называем повара, имя не влияет на тесты. В моем варианте - это Amigo :)

Сверим выводы в консоль. Пример моего вывода:

Your order: [Water] of Tablet{number=5}

Start cooking - Your order: [Water] of Tablet{number=5}

Your order: [Water] of Tablet{number=5} was cooked by Amigo

Требования:  
1. Класс Waiter должен реализовывать интерфейс Observer.  
2. Класс Cook должен быть потомком класса Observable.  
3. В методе update класса Cook должен быть вызван метод setChanged.  
4. В методе update класса Cook должен быть вызван метод notifyObservers с текущим заказом в качестве параметра.  
5. Метод update в классе Waiter должен выводить на экран строку форматированную согласно заданию.  
6. В методе main класса Restaurant должен быть создан новый официант и добавлен повару в качестве наблюдателя с помощью метода addObserver.