**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.

### Ресторан(5)

Сейчас повар готовит блюда моментально. На самом деле приготовление занимает какое-то время.  
Нам нужно вычислить время приготовления всего заказа, чтобы потом подбирать рекламные ролики для этого времени.

1. Предположим, что нам известно время приготовления каждого блюда в минутах. Захардкодим его в классе **Dish**.  
1.1. Измени создание элементов **enum** — Fish(25), Steak(30), Soup(15), Juice(5), Water(3);  
1.2. Создай поле private int duration с **геттером**.  
Чтобы создать **геттер**, нажмите Alt+Insert и выберите пункт **Getter**. Далее выберите имя поля и нажмите **OK(Enter)**.

2. Добавим возможность посчитать продолжительность приготовления всего заказа.  
Куда его добавить???  
2.1. Если в класс **Cook**, то повар сможет считать продолжительность приготовления заказа.  
Чтобы другие классы могли получить эту информацию, необходимо будет получить доступ к объекту **Cook**.  
2.2. Если в класс **Order**, то имея доступ к заказу всегда можно узнать продолжительность приготовления.  
2.3. Выбери правильное место из **п.2.1.** и **п.2.2.** и добавьте метод pubic int getTotalCookingTime(), который посчитает суммарное время приготовления всех блюд в заказе.  
2.4. Добавим нашему повару вывод в консоль этой информации. Пусть теперь выводится аналогичное сообщение:  
Start cooking — Your order: [Soup, Juice, Water] of Tablet{number=5}, cooking time 23min

Наведем красоту:  
3. Запустим приложение и сразу введем ‘**exit**‘. Вывод получился не очень красивым.  
Сделай так, чтобы если в заказе нет блюд, он не отправлялся повару. Найди это место и реализуйте логику.  
В классе **Order** создай вспомогательный метод public boolean isEmpty(), который будет определять, есть ли какие либо блюда в заказе.

Требования:  
1. Время приготовления блюд должно соответствовать условию задачи.  
2. Поле duration в enum Dish должно быть приватным и иметь публичный геттер.  
3. В классе Order должен быть создан метод public int getTotalCookingTime возвращающий количество времени требующееся на приготовление текущего заказа.  
4. Помимо списка блюд в заказе, на экран также должно быть выведено время приготовления заказа.  
5. Если в заказе отсутствуют блюда, он не должен быть отправлен повару.  
6. Метод isEmpty в классе Order должен возвращать true, если заказ пуст.

### Ресторан(6)

Заказ создается, потом готовится поваром, а после этого относится посетителю. К тому же считается время выполнения заказа.  
Будем считать, что первая часть задания выполнена.  
Перейдем ко второй — пока заказ готовится, на планшете должна показываться реклама.  
Определим, что такое реклама.

Реклама — это видео определенной продолжительности. Также известно, что кто-то оплатил количество показов.  
Будем считать, что у нас известно количество оплаченных показов, общая стоимость всех показов и сам рекламный ролик.

1. Для связанного с рекламой функционала создадим пакет **ad**.

2. Создадим класс **Advertisement**(Рекламное объявление) в пакете **ad**, у которого будут следующие поля:  
Object content — видео  
String name — имя/название  
long initialAmount — начальная сумма, стоимость рекламы в копейках. Используем long, чтобы избежать проблем с округлением  
int hits — количество оплаченных показов  
int duration — продолжительность в секундах  
Модификаторы доступа расставь самостоятельно.

3. В классе **Advertisement** создадим конструктор с порядком аргументов, соответствующим перечисленной последовательности всех полей класса

4. В этом же пакете создадим еще два класса:  
AdvertisementStorage — хранилище рекламных роликов.  
**AdvertisementManager** — у каждого планшета будет свой объект менеджера, который будет подбирать оптимальный набор роликов и их последовательность для каждого заказа.  
Он также будет взаимодействовать с плеером и отображать ролики.

5. Так как хранилище рекламных роликов **AdvertisementStorage** единственное для всего ресторана, то сделаем его синглтоном.

Требования:  
1. В пакете ad должен быть создан класс Advertisement.  
2. В пакете ad должен быть создан класс AdvertisementManager.  
3. В пакете ad должен быть создан класс AdvertisementStorage.  
4. В классе Advertisement в пакете ad должны быть объявлены все поля перечисленные в условии задачи.  
5. В классе Advertisement в пакете ad должен быть создан корректный конструктор в соответствии с условием задачи.  
6. Класс AdvertisementStorage должен быть синглтоном.

### Ресторан(7)

При выборе, с какого класса начать, старайся найти тот, который используют другие, и который сам ничего не использует.  
В нашем случае — это хранилище рекламных роликов (**AdvertisementStorage**).  
Мы решили, что оно будет единственным в ресторане и сделали его синглтоном.

Опишем его.  
1. Видео должно где-то храниться, пусть это будет список.  
Создадим поле videos и инициализируем его пустым листом.  
Подумай, должно ли поле videos иметь возможность менять свое значение?

2. Чтобы как-то работать с видео, создай публичные методы:  
2.1. list() — который вернет список всех существующих доступных видео.  
2.2. add(Advertisement **advertisement**) — который добавит новое видео в список **videos**.

3. В конструкторе класса добавим в список videos какие-то данные. У меня это:  
Object someContent = new Object();  
new Advertisement(someContent, «First Video», 5000, 100, 3 \* 60) // 3 min  
new Advertisement(someContent, «Second Video», 100, 10, 15 \* 60) //15 min  
new Advertisement(someContent, «Third Video», 400, 2, 10 \* 60) //10 min

4. В AdvertisementManager создадим **final** поле-ссылку на экземпляр AdvertisementStorageи назовем ее **storage**.  
Не забудь инициализировать созданное поле!

Требования:  
1. В классе AdvertisementManager должно быть создано поле storage содержащее экземпляр класса AdvertisementStorage.  
2. В классе AdvertisementStorage список videos должен быть объявлен с модификатором доступа final.  
3. В классе AdvertisementStorage метод list должен возвращать список videos и быть публичным.  
4. В классе AdvertisementStorage метод add должен принимать один параметр типа Advertisement и добавлять полученный объект в список videos.  
5. В конструкторе класса AdvertisementStorage в список videos должно быть добавлено несколько объектов типа Advertisement.

### Ресторан(8)

Пришло время описать функционал класса **AdvertisementManager**.  
Но для начала нам понадобятся некоторые методы в классе **Advertisement**.

1. В классе **Advertisement** создай поле long amountPerOneDisplaying.  
Оно должно равняться стоимости одного показа рекламного объявления в копейках (**initialAmount/hits**).  
Присвой значение полю в конструкторе.

2. В классе **Advertisement** создай геттеры для полей name, duration и amountPerOneDisplaying.

3. Определим необходимые данные для объекта AdvertisementManager — это время выполнения заказа поваром.  
Т.к. продолжительность видео у нас хранится в секундах, то и и время выполнения заказа тоже будем принимать в секундах.  
В классе AdvertisementManager создай конструктор, который принимает один параметр — int timeSeconds.  
Создай соответствующее поле и сохраните это значение в него.

4. AdvertisementManager выполняет только одно единственное действие — обрабатывает рекламное видео.  
Поэтому создайте единственный публичный метод void processVideos(), его функционал опишем в следующем задании.  
А пока выведем в консоль «***calling processVideos method***»

5. Чтобы тестировать данную функциональность, нужно добавить вызов processVideosметода у AdvertisementManager.  
Очевидно, что этот метод должен вызываться во время создания заказа, а точнее — в параллельном режиме.  
Заказ готовится в то время, как видео смотрится.  
Добавьте вызов метода processVideos() в нужное место.

**P.S.** Не забудь что время приготовления заказа считается в минутах, а время показа рекламы в секундах!

Требования:  
1. В классе Advertisement должны быть созданы корректные геттеры для полей name, duration и amountPerOneDisplaying.  
2. В классе Advertisement должно быть создано поле amountPerOneDisplaying равное частному от деления initialAmount на hits.  
3. В классе AdvertisementManager должен быть создан конструктор с одним параметром типа int инициализирующий поле timeSeconds.  
4. В классе AdvertisementManager должен быть создан метод processVideos, оповещающий о своем вызове выводом строки на консоль.  
5. В методе createOrder класса Tablet должен быть создан новый AdvertisementManager и у него должен быть вызван метод processVideos.

### Ресторан(9)

Нам понадобится исключение, которое поможет обработать ситуацию, если у нас не будет получаться подобрать рекламные ролики.

1. Создадим **unchecked** исключение NoVideoAvailableException в пакете **ad**.

2. Разберем подробно метод void processVideos() в AdvertisementManager.  
2.1. Удаляем из него вывод в консоль «***calling processVideos method***»  
**Метод должен:**  
2.2. Подобрать список видео из доступных, просмотр которых обеспечивает максимальную выгоду. (***Пока делать не нужно, сделаем позже).***  
2.3. Если нет рекламных видео, которые можно показать посетителю, то бросить **NoVideoAvailableException**, которое перехватить в оптимальном месте (***подумать, где это место***) и с уровнем Level.INFO логировать фразу «***No video is available for the order***» ***+*** order  
2.4. Отобразить все рекламные ролики, отобранные для показа, в порядке уменьшения стоимости показа одного рекламного ролика в копейках. Вторичная сортировка — по увеличению стоимости показа одной секунды рекламного ролика в тысячных частях копейки.  
Используйте метод Collections.sort  
(***Тоже пока делать не нужно, сделаем позже***).

**Пример для заказа** [Water]**:**  
First Video is displaying... 50, 277  
где **First Video** — название рекламного ролика  
где **50** — стоимость показа одного рекламного ролика в копейках  
где **277** — стоимость показа одной секунды рекламного ролика в тысячных частях копейки (равно **0.277** коп)  
Используйте методы из класса **Advertisement**.  
2.5. В классе **Advertisement** создайте метод void revalidate(). Этот метод должен:  
2.5.1. Бросать **UnsupportedOperationException**, если количество показов не положительное число.  
2.5.2. Уменьшать количество показов.

Требования:  
1. В случае, если список видео для воспроизведения пуст, должно быть брошено исключение NoVideoAvailableException из метода processVideos.  
2. Исключение NoVideoAvailableException должно быть перехвачено и в лог должна быть записана строка об отсутствии видео для данного заказа (пример в условии).  
3. Класс NoVideoAvailableException в пакете ad должен быть потомком класса RuntimeException.  
4. Метод revalidate должен быть реализован в соответствии с условием задачи.

**Ресторан(10)**

Рекурсию используют тогда, когда алгоритм решения задачи совпадает с алгоритмом решения подзадачи (*части*).  
У нас как раз такой случай. Нам нужно сделать полный перебор всех вариантов и выбрать из них лучший.

Напомню, рекурсия пишется по следующему принципу:  
а) условие выхода/окончания рекурсии  
б) условие продолжения — вызов самой себя с набором параметров предыдущего шага.  
В любое время ты можешь почитать в инете подробную информацию по написанию рекурсии.

Текущее задание — реализовать п.2.2 предыдущего задания с помощью рекурсии.  
(*подобрать список видео из доступных, просмотр которых обеспечивает максимальную выгоду*)  
Рекурсивный метод должен выбрать набор рекламных роликов, которые будут показаны посетителю.

Этот набор должен удовлетворять следующим требованиям:  
1. сумма денег, полученная от показов, должна быть максимальной из всех возможных вариантов  
2. общее время показа рекламных роликов НЕ должно превышать время приготовления блюд для текущего заказа;  
3. для одного заказа любой видео-ролик показывается не более одного раза;  
4. если существует несколько вариантов набора видео-роликов с одинаковой суммой денег, полученной от показов, то:  
4.1. выбрать тот вариант, у которого суммарное время максимальное;  
4.2. если суммарное время у этих вариантов одинаковое, то выбрать вариант с минимальным количеством роликов;  
5. количество показов у любого рекламного ролика из набора — положительное число.

При локальном тестировании учитывайте, что необходимо отобразить все рекламные ролики, отобранные для показа, в порядке уменьшения стоимости показа одного рекламного ролика в копейках.

Также не забудь реализовать п.2.4 из предыдущего задания (*вывести на экран все подходящие ролики)*.

Для каждого показанного видео-ролика должен быть вызван метод revalidate().

Требования:  
1. Сумма денег, полученная от показов, должна быть максимальной из всех возможных вариантов.  
2. Общее время показа рекламных роликов НЕ должно превышать время приготовления блюд для текущего заказа.  
3. Для одного заказа любой видео-ролик должен показываться не более одного раза.  
4. Если существует несколько вариантов набора видео-роликов с одинаковой суммой денег, полученной от показов, то должен быть выбран вариант с максимальным суммарным временем.  
5. Если существует несколько вариантов набора видео-роликов с одинаковой суммой денег и одинаковым суммарным временем, то должен быть выбран вариант с минимальным количеством роликов.  
6. Для каждого показанного ролика должен быть вызван метод revalidate.