Programación II

Ejercicio: Gestión VIP para venta de entradas curso 23/24

Dpto. LSIIS. Unidad de Programación

1 Introducción

En este ejercicio se van a trabajar los siguientes conceptos:

1. Creación de una clase genérica
2. Uso de colas
3. Uso de un ArrayList de colas
4. Aplicar herencia para conseguir una clase que sea idónea para el problema a resolver partiendo de otra más general

2 El problema

El problema que se desea resolver consiste en la gestión de espera de los clientes que desean comprar las entradas para un evento. La solución sólo se encarga de registrar a los clientes y de sacarlos en el orden establecido. El orden viene determinado por la prioridad del cliente, es decir como de importante es.

El sistema tendrá varios niveles de importancia de 1 a N siendo N el nivel más importante y 1 el menos.

La operativa del sistema es el siguiente. Cuando llega la solicitud de un cliente para comprar entradas, si no hay nadie esperando en el sistema, se le atiende. En el caso de que hubiera alguien esperando en el sistema, se le coloca en la cola que corresponda a su prioridad.

2.1 Selección del siguiente a ser atendido

El criterio para elegir a un cliente será el siguiente:

1. Se coge al cliente de mayor prioridad disponible que lleve más tiempo esperando, siempre y cuando se cumpla el criterio expuesto en el siguiente punto.
2. Para evitar que haya colas que no avancen nunca se establece el siguiente criterio de cesión de turno, que permite avanzar a las colas de menor prioridad:
    a. Si de una cola se han atendido a cinco o más clientes, se seleccionará a un cliente de inferior prioridad, si lo hubiera. El contador de la cola que ha cedido el turno se pondrá a cero.
    b. Si a una cola le corresponde ceder el turno, pero no hay elementos esperando en las prioridades inferiores, el contador de solicitudes atendidas en esta prioridad se pone a cero y se atiende al individuo que

lleve más tiempo esperando en la cola de la prioridad que le correspondía ceder el turno.

c. Si cuando una prioridad cede el turno y la siguiente prioridad que tiene solicitudes pendientes está en situación de ceder el turno esta se cederá a la de siguiente prioridad disponible. En el caso de no haber nadie esperando en las prioridades inferiores se aplica lo estipulado en el apartado **b.**

2.2 Lo que se pide

Se pide diseñar un TAD adecuado para resolver este problema es decir una cola de prioridades

3 Pasos a seguir para resolver el problema

3.1 Cola de prioridades con política de extracción simple

Lo primero que se debe plantear es el resolver el problema de una cola de prioridades en donde el siguiente elemento a atender sea el que lleva más tiempo esperando en la cola de mayor prioridad con elementos por ser atendidos. Esta clase que llamaremos PriorityQueue y estará en el paquete priorityQueue. Los servicios que debe tener esta cola son:

- Constructor: Recibe el número de prioridades que debe tener
- add: Recibe un entero entre uno y N donde N es el número de prioridades y el dato que se desea insertar en la cola correspondiente. En el caso de que no se cumpla la precondición se generará la excepción: IncorrectPriority.
- getTotalEsperando: Método que retorna el número total de elementos esperando
- isEmpty: Método que retorna cierto si la cola está vacía.
- getNext: Método que retorna el siguiente elemento que tiene que ser atendido. El orden en el que se selecciona el siguiente elemento es el de retornar el elemento que lleva más tiempo esperando en la cola de mayor prioridad. La cola no debe estar vacía. En el caso de que no se cumpla la precondición se debe lanzar la siguiente excepción EmptyQueueException

---

No seguir leyendo hasta terminar lo anterior

3.2 Cola de prioridades necesaria para nuestro ejercicio

Una vez se tiene desarrollada y probada se procede a la siguiente fase en la solución de nuestro problema.

Ahora se dispone de una pieza más para resolver nuestro problema. Para aprovechar esa pieza vamos a utilizar otro recurso de la POO, **la herencia**. La diferencia entre nuestra cola y la que hemos implementado (más general) es la política para obtener al siguiente. Es decir, todo lo demás nos serviría (posiblemente tengamos que realizar cambios en la clase que acabamos de desarrollar para poder aprovecharla añadiendo métodos protegidos que permitan a la clase hija hacer su trabajo)

Los servicios que debe proporcionar son los mismos salvo que se ha de reemplazar lo que hace getNext, es decir se tendrá que sobrescribir