Programación Concurrente y Distribuida

- Proyecto Puerto Marítimo -

Supongamos que estamos programando un sistema en el que, de momento, queremos controlar las entradas y salidas de barcos de un Puerto. Tenemos dos tipos de barcos: los que quieren entrar y los que quieren salir y una puerta de control por la que entran y salen los barcos.

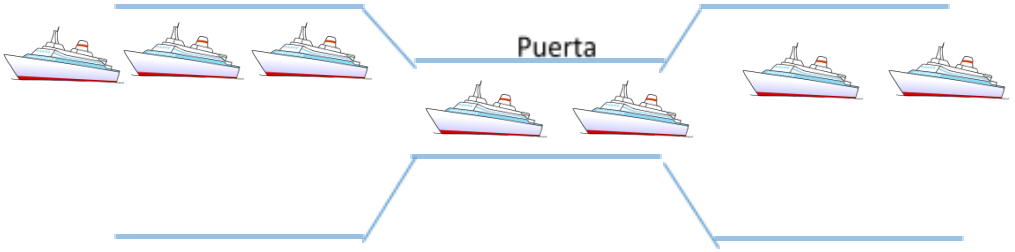

Figura 1 Paso 1: Barcos entrando y saliendo del Muelle (en este caso colisionan 2 barcos en la puerta)

Esta práctica la vamos a ir haciendo de forma incremental. En cada paso haremos modificaciones que nos ayuden a entender algunos de los conceptos que vamos introduciendo en las clases de teoría. A cada modificación la llamaremos *paso*.

Paso 1.-

Crearemos un proyecto denominado `Barcos`.

Crear una clase *Puerta* que implemente los métodos `entrar` y `salir`. Crear una clase `Barco` que llama al método `salir` o `entrar` de la clase `Puerta` (dependiendo de si es barco de entrada o de salida).

El hecho de entrar o salir lo simularemos imprimiendo 3 veces el mensaje "El barco X entra/sale", siendo *X* el `id` del barco que recibe por parámetro en el constructor. El barco también recibe otro parámetro que indica si es un barco que sale o que entra al Puerto.

Crear un programa principal que lance diversos threads `Barco`, unos que quieran *entrar* y otros que quieran *salir* (por ejemplo, 10 de entrada y 10 de salida).

En esta solución no te preocupes si los barcos colisionan en la `Puerta` (Figura 1) Esto ocurrirá si los mensajes se entremezclan de forma no deseada.

A continuación, tienes dos extractos de posibles salidas, en la primera no están colisionando y en la segunda sí.

No hay colisión	**Sí hay colisión**
El barco 1 entra	El barco 1 entra
El barco 1 entra	El barco 2 sale
El barco 1 entra	El barco 2 sale
El barco 2 sale	El barco 1 entra
El barco 2 sale	El barco 1 entra
El barco 2 sale	El barco 2 sale

Un posible diagrama de clases de la solución lo tienes en la Figura 2. No tienes por qué seguir exactamente este diagrama. Simplemente es una ayuda.

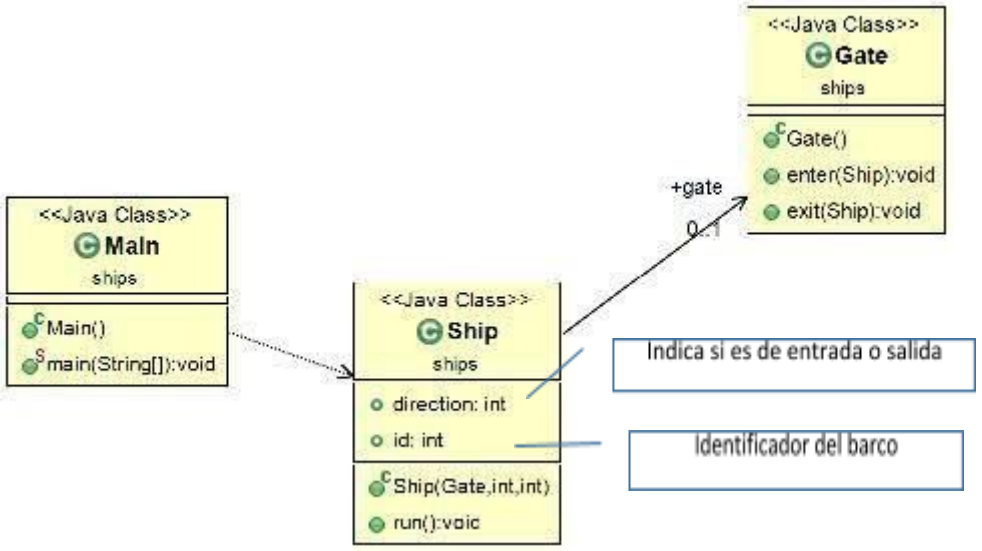

Figura 2 Diagrama de clases para el Paso 1

Paso 2.-

Modificar el programa para que sólo pase un barco a la vez por la puerta de control. Esto implica hacer uso de `synchronized`.

Paso 3.-

Ahora pueden pasar por la puerta de control más barcos, siempre y cuando todos vayan en el mismo sentido. Si quiero entrar y hay otro entrando en el mismo sentido, podré hacerlo. Si no está pasando nadie, también podré pasar. Si viene alguien de frente no. Igual para las Salidas.

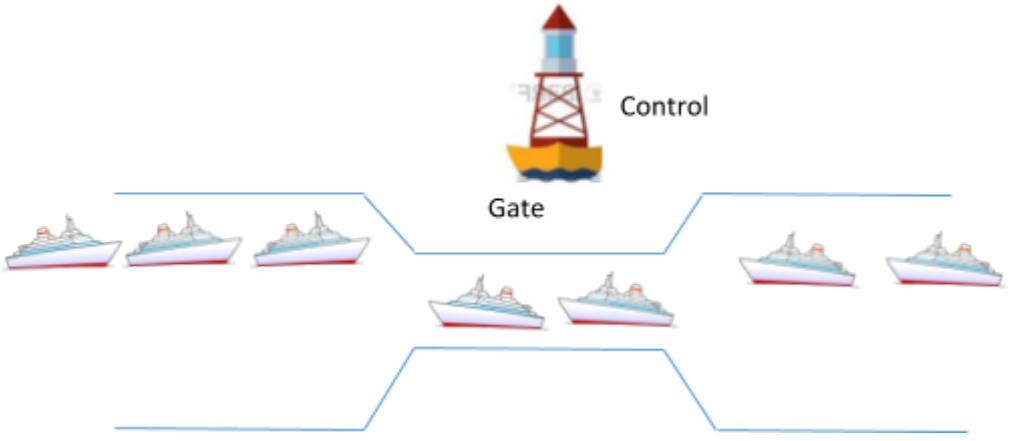

Figura 4 Paso 3: Varios barcos pasando en el mismo sentido

Puesto que esto se va a ir complicando poco a poco, vamos a optar por crear una nueva clase denominada `TorreDeControl` que es la encargada de gestionar los permisos de entrada y salida por la puerta. Tendrá los siguientes métodos: `permisoEntrada`, `permisoSalida`, `finEntrada` y `finSalida`. Ahora, antes de entrar, se invocará el método `permisoEntrada`

de `TorreDeControl`. Cuando termina de entrar, se invocará el método `finEntrada` para que la torre sepa que ha terminado de entrar y haga las acciones oportunas. En la Figura 5 tienes el diagrama de clases de la solución propuesta. Una vez más indicar que no tienes por qué seguir exactamente este diagrama de clases, puedes optar por otro. Simplemente es una guía que es recomendable seguir.

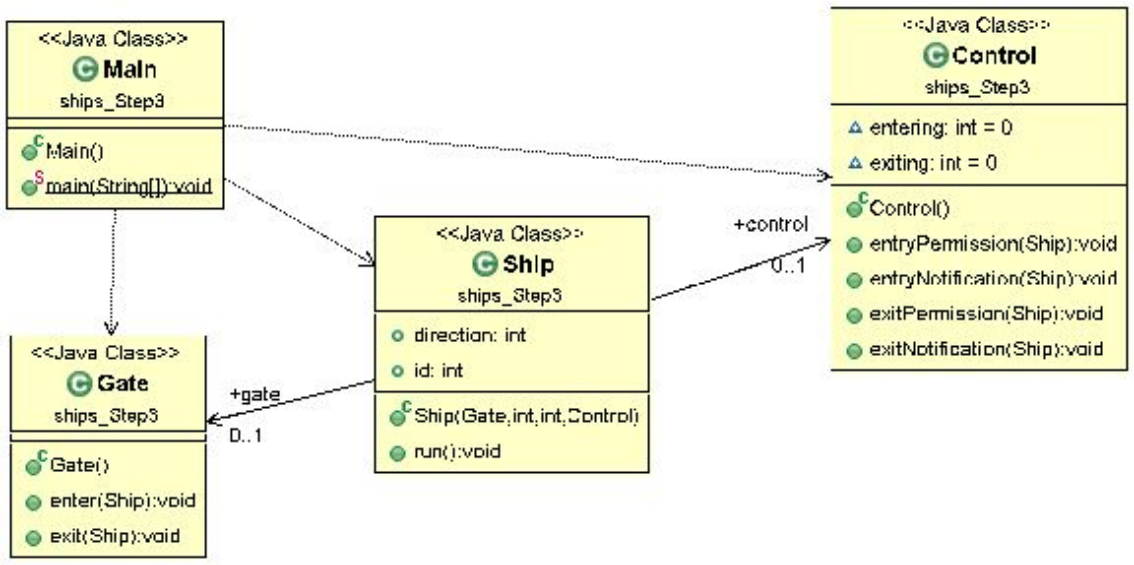

Figura 5 Paso 3: Diagrama de clases para el paso 3

Paso 4.-

Además de lo anterior, ahora tienen preferencia los que quieren salir. Si hay algún barco que quiere entrar, pero hay algún barco que está esperando por salir, entonces no puede entrar. Sólo lo hará cuando no haya nadie saliendo ni esperando por salir.