Trabajo Práctico Obligatorio Final

PARQUE “ECO-PCS”

Programación Concurrente

- *Alumna:* Natalia Belén Narváez
- *Legajo:* FAI-3198

- *Año de Cursado:* 2do cuatrimestre del 2022

Este informe tiene como finalidad dar explicación a las soluciones implementadas junto al porqué de los mecanismos de sincronización elegidos.

Inicio de la simulación del funcionamiento del parque ecológico "ECO-PCS":

- Clases relacionadas: Colectivo.java, Colectivero.java

Los visitantes tienen la opción de llegar al parque en forma particular o por tour, cuando elige la primera opción pasa directamente a la elección de la actividad deseada. Si no, al optar por el tour a través de colectivos folklóricos debe esperar por el mismo. Para esto elegí como mecanismo de sincronización a los cerrojos/locks ya que me permiten simular la espera a un colectivo, el recorrido que realiza y la terminación.

El lock tendrá como una de sus condiciones asociadas "esperaColectivo" donde los hilos visitantes se quedarán esperando en caso de que se haya llenado su capacidad máxima (25 personas) o el recorrido ya haya comenzado. A su vez, estará la condición "cole" en donde los visitantes que ya subieron al colectivo quedarán bloqueados hasta que termine el recorrido. Finalmente, estará la condición "colectivero", en la que el hilo Colectivero esperará que se complete la capacidad del colectivo o, al tener un await(3, TimeUnit.SECONDS), a los 3 segundos de esperar iniciará el tour. Es dicho hilo quien se encargará de terminar el recorrido y avisarles a los visitantes que ya pueden bajarse como también despertar a aquellos que hayan quedado esperando subir. Este ciclo continúa hasta que cierra el ingreso al parque. Una vez ha cerrado, el hilo Colectivero terminará su ejecución y los hilos visitantes que hubieran quedado esperando por el tour, son avisados y se retiran del parque.

Actividades dentro del parque:

- *Shop de souvenirs:*
  - Clases relacionadas: Shop.java

Decidí utilizar semáforos genéricos debido a que, al necesitar simular dos cajas por las cuales poder pagar, me permitía reducir los tiempos de espera de cada hilo visitante a la hora de querer pagar. Además, simularía la espera habitual de la caja de un supermercado en la vida real, donde se hace una única fila y se nos indica a que caja pasar ni bien se desocupa alguna.

Si el hilo visitante elige ir al shop, primeramente, adquirirá un souvenir dependiendo de si el parque continúa o no abierto. En caso positivo, hará un sleep de 3 segundos para simular el tiempo de elección y luego se trabará en el semáforo "cajas" hasta que consiga el permiso y así pueda avanzar a pagar.

Hay otro semáforo "tienda", el cual se utiliza para realizar los chequeos de si sigue abierto o no el parque, y por consiguiente si pueden o no entrar los hilos visitantes. También me permite que el controlador del parque se encargue de cerrar la tienda correctamente, esperando el permiso y cambiando de valor la variable sigueAbierto (la cual indica el estado del parque).

- *Snorkel:*
  - Clases relacionadas: Snorkel.java

Para la actividad de snorkel seleccioné como método de sincronización monitores, puesto que facilitaba la manera de comunicarse entre hilos a la hora de avisar si devolvieron un equipo y únicamente necesitaba una zona de espera.

Al entrar a la actividad de snorkel, si no hay equipos disponibles (compuesto por snorkel, salvavidas y patas de rana, pero representado por una única variable "equiposDeSnorkel") el visitante deberá esperar a que se libere uno. Al conseguirlo, hará un sleep de 3 segundos para simular el tiempo de sumergida. Una vez ha terminado de hacer snorkel, devuelve el equipo y avisa al resto de hilos que se ha liberado uno. Si el parque ha cerrado mientras esperan por un equipo, al ser notificados, se retirarán del parque.

- *Restaurantes:*
  - Clases relacionadas: Restaurante.java

Como indicaba que cada restaurante tiene habilitada una cola de espera cuando se superó su disponibilidad, consideré que la mejor opción sería utilizar una Blocking Queue porque simularía una fila por orden de llegada y únicamente podrían continuar su ejecución una vez que entraron. Será llamada "restaurante" y tendrá asignado como tamaño del arreglo la capacidad del restaurante (diferente por cada uno de los 3 restaurantes).

En caso de que pudieran escribir dentro de la cola bloqueante, continuarán con un sleep de 6 segundos si almorzarán y 5 si merendarán, indicando en su respectiva variable booleana que ya

almorzaron/merendaron (dado que sólo pueden hacer cada acción una única vez). Luego resta salir del restaurante, donde harán un take sobre “restaurante”, liberando así un cupo para que entre el siguiente visitante que estaba esperando. Si el parque ha cerrado mientras esperan ingresar, al intentar entrar serán notificados y se retirarán del parque.

- *Faro-Mirador:*
  - Clases relacionadas: Faro.java, AdminTobogan.java

Elegí representar el Faro-Mirador con cerrojos/locks dado que necesitaba tener múltiples conjuntos de espera y además poder hacer que se respete el orden de llegada. Se verán involucrados en esta simulación los hilos visitantes y un hilo adminTobogan, que será el encargado de asignar los toboganes.

Para acceder al tobogán es necesario subir por una escalera caracol, que tiene capacidad para n personas. Ésto lo controlo con una condición “esperaSubir”, en la cual se quedarán esperando debajo de la escalera hasta que se desocupe un espacio arriba de la misma. Si son el primero en subir dan aviso al adminTobogan (con una condición con el mismo nombre) para que despierte. Luego, entran a la condición “esperaTirarse”, de la cual no saldrán hasta que el adminTobogan no les asigne un tobogán por el cual tirarse. Si se les asigna un tobogán, el mismo ya estará desocupado por lo que continuarán su ejecución y se tirarán, simulado con un sleep de 2 segundos. Al despertar avisarán al adminTobogan que el tobogán por el cual se tiraron está disponible nuevamente y se retirarán.

El adminTobogan estará ejecutando un único método hasta el cierre del parque, en donde asignará constantemente un tobogán a cada visitante listo para tirarse. En un principio, al no haber nadie en la escalera se queda en la condición “adminTobogan” esperando que llegue algún visitante. Luego, si ninguno de los toboganes está libre espera en “hayToboganLibre” hasta que le notifiquen que alguno se desocupó, verifica cuál tobogán está disponible, lo marca como no libre y lo asigna a la variable “toboganAsignado” (que es la que esperan los visitantes para tirarse), notifica al primer visitante listo, resta uno a la cantidad de visitantes esperando en la escalera (ya que se acaba de tirar el primero) y también avisa a quien está esperando subir a la escalera para que pueda ponerse en la fila para tirarse.

- *Carrera de gomones por el río:*
  - Clases relacionadas: CarreraGomones.java, AdminCarreraGomones.java, Gomon.java, StandDeBicis.java, Tren.java, Maquinista.java

Los visitantes tienen la posibilidad de llegar al inicio de la actividad a través de bicicletas o un tren interno con capacidad de máximo 15 personas. El stand de bicicletas permite que un visitante utilice una bicicleta únicamente si hay disponibilidad, en caso contrario deberá esperar en la condición biciDisponible. Mientras el parque esté abierto podrá viajar en bicicleta. Si pudo subir a una tardará entre 4 y 7 segundos (simulado con un sleep y un random entre dichos números) y luego, al llegar a la carrera de gomones, la dejará y dará aviso de que hay una bicicleta desocupada. En el caso del tren, su funcionamiento es símil al del colectivo para tours folklóricos, teniendo una condición llamada "esperaTren" en la cual quedará bloqueado si se superó la capacidad máxima o el recorrido ya comenzó. Una vez puede subir, esperará el fin del viaje en la condición "tren" y será el último en bajar quien indique a los hilos esperando subir que pueden hacerlo. El hilo maquinista esperará 7 segundos, o menos si se completa antes, para iniciar el recorrido y dormirá por 5 segundos simulando el viaje en tren. Tras despertarse avisará a los visitantes de que pueden bajar y modifica la variable empezó a falso, que indica si el recorrido esta en curso o no.

Como la largada necesita que haya una cantidad de gomones determinada, sin importar el tipo de los mismos, decidí utilizar una Cyclic Barrier con esa cantidad. Los visitantes decidirán si quieren un gomon individual o doble. Para la espera de los gomones por parte de los visitantes y la simulación de la carrera utilicé locks ya que necesitaba tener áreas de espera dependiendo de varias condiciones. También incluí semáforos en el control de los hilos gomones y el hilo adminCarreraGomones, principalmente para la comunicación uno-a-uno entre hilos.

Al pedir un gomon individual, primeramente, el hilo visitante deberá esperar en "esperaIndiv" si los gomones individuales ya están todos en uso. Luego, si los gomones individuales sumado a los dobles en uso supera la cantidad necesaria para la largada o ésta ya comenzó, deberá esperar a la próxima largada en "esperaProximaLargada". Una vez pasó las condiciones necesarias podrá subir a un gomon, aumentará la variable "individualesEnUso" y llamará a un gomon individual mediante el semáforo "semIndividual". Posteriormente, se aumenta la variable que controla la cantidad de visitantes que tienen que estar presentes para la salida llamada "totalVisitParaCarrera",

se avisa al adminCarreraGomones por si llega a ser el último y finalmente queda esperando la carrera en la condición “largada”.

Por otra parte, al pedir un gomon doble se realiza un recorrido similar hasta la espera de la próxima largada. Al pasar las condiciones necesarias aumento la variable “esperandoCompa”, que me permite saber si es el primer o segundo hilo en solicitar uno. En caso de ser el primero, deberá esperar en la condición “esperaCompa” hasta ser despertado por un segundo hilo que quiera correr junto a él. Si es el segundo, esperandoCompa pasa a valer 0 (no hay nadie esperando), incrementa la variable “doblesEnUso”, despierta al hilo en “esperaCompa” y avisa a un gomon doble. Finalmente, el visitante que solicitó un gomon doble incrementa el “totalVisitParaCarrera”, por si llegara a ser el último avisa a “esperaAdminCarrera” y espera en la condición “largada” hasta que termine la carrera.

Cuando termina la carrera los visitantes ejecutan un método que indica que pidieron sus pertenencias al final del recorrido.

Los hilos gomones estarán ciclando hasta el cierre del parque. Si son de tipo individual irán al método esperaIndiv(), mientras que si son de tipo doble a esperaDoble(). Ambos métodos harán que el gomon quede bloqueado en un semáforo semIndividual o semDoble (respectivamente), y serán despertados cuando un visitante pedir uno. Posteriormente ejecutarán carrera(tipo), donde deberán conseguir un permiso del semáforo controlLargada(gomonesParaLargada, true) que sólo permitirá que pasen los primeros gomones y que no lleguen a la barrera cíclica antes de que termine la carrera actual, además de no dejarlos largar sin que estén listos (se hayan despertado) todos los hilos visitantes que van a participar de la carrera. Luego pararán en la barrera cíclica “salida” y al completarse la cantidad necesaria comenzará la carrera. Dormirán entre 3 y 4 segundos para simular el tiempo de bajada y finalmente ejecutarán terminarCarrera(tipo). Aquí, dependiendo de su tipo restarán uno a las variables que llevan cuenta de los gomones en uso y se indicará en que posición terminaron. Siendo el último quien avise mediante el semáforo “semAdminCarrera” al adminCarreraGomones que ha terminado la carrera.

El adminCarreraGomones también estará ciclando hasta el cierre del parque. Ejecutará en primera instancia el método iniciarCarrera(), donde quedará esperando en la condición “esperaAdminCarrera” hasta que haya como mínimo la misma cantidad de visitantes listos para la

carrera como gomones necesarios (por ejemplo, que haya capacidad 7 y debe haber, aunque sea, 7 visitantes subidos) o sino que esa cantidad de visitantes sea igual a la suma de los gomones individuales y dobles en uso (multiplicando por 2 a la variable de los dobles ya que cuenta por cada visitante). Luego, reinicia totalVisitParaCarrera (para la próxima largada) y libera la cantidad total de permisos del semáforo "controlLargada", que depende de cuantos gomones hacen falta para salir. Luego, ejecutará esperaFinalizarCarrera() en el que se bloqueará en el semáforo "semAdminCarrera" hasta que el último gomón en llegar le de el aviso de que terminó la carrera. Finalmente efectuará finalizarCarrera() realizando el aviso de la condición "largada" para que los hilos se bajen de los gomones y se retiren de la actividad, como también indicando a los hilos que esperan la próxima largada, un gomon individual o doble que despierten.

Cierre y control del parque:

- Clases relacionadas: Parque.java, ControladorParque.java

El funcionamiento del parque será determinado con locks. Dado que el complejo se encuentra abierto para el ingreso de 09:00 a 17:00hrs y que las actividades cierran a las 18:00hrs, cree un hilo ControladorParque que tendrá la responsabilidad de abrir, cerrar el ingreso y las actividades del parque. Para ésto ejecutará tres métodos de la clase Parque en el siguiente orden; abrirParque(), donde pondrá en verdadero las variables "abierto" y "comenzarActividad" (que son sobre las que pregunta el visitante si puede entrar como también cada vez que esta por repetir la elección de una actividad) además dará aviso a todos los visitantes que estén en la condición "esperaApertura". Luego será el turno de cerrarIngresoParque(), aquí quedará bloqueado en la condición "controlador" durante 15 segundos, pasado este tiempo indicará que ya no se puede ingresar al parque y cambiará la variable abierto a false. Finalmente pasará a cerrarActividades(), esperará 10 segundos extras y empezará con el cierre total del parque. Cambiará la variable comenzarActividad a falso (así los visitantes saldrán del while que los hace elegir actividades) y hará un signal a aquellos hilos que quedaron en la condición "esperaApertura" para que se retiren. Pasará a cerrar todas las actividades del parque con shop.cerrarShop(), snorkell.cerrarSnorkel(), restaurantes[0 a 2].cerrarRestaurante(), faro.cerrarFaro(), carreraGomones.cerrarCarrera(), standBicis.cerrarStandBicis(), tren.cerrarTren(). En cada uno de estos métodos, que pertenecen a la actividad correspondiente, hace un cambio a la variable "sigueAbierto" que es la que verifican a la hora de entrar a una actividad.

Algunas aclaraciones extras:

- Clases relacionadas: ColoresSout.java, Main.java

La clase ColoresSout la cree para poder dar color a los diferentes mensajes que salen en la terminal, con el fin de que sea un poco mas claro el seguimiento de las acciones de los hilos en ejecución. Busqué en internet como realizarlo y así obtuve los valores de cada color.

En el main se puede asignar la cantidad de visitantes que entran al parque, la variable cantVisitantes1 es la primera “ola” en llegar, mientras que cantVisitantes2 son los que arriban luego de un tiempo. Hago que el hilo main duerma en dos ocasiones, así puedo simular que los visitantes llegan en diferentes tandas y así observar cómo interactúan con el parque una vez ha cerrado su ingreso o llegando mas tarde que los iniciales.