**Infraestructura Computacional – Sección 1**

**Profesor: Harold Castro**

**Informe - Caso 1**

**Manejo de la concurrencia**

* **Esteban Ortiz - 201913613**
* **Sergio Guillen - 201912757**

**Funcionamiento**

Para el caso planteado se implementó una arquitectura de Productores y Consumidores con 6 clases y un archivo properties.txt detallados a continuación.

* **Main –** Esta clase se encarga de crear tanto el buzón intermedio, el buzón productor y el buzón consumidores. También comienza la ejecución de los threads Productores, Consumidores e Intermedios. Crea y ejecuta una nueva carga y toma la cantidad de productores y consumidores.
* **Carga –** Esta clase se encarga de leer el archivo properties.txt. Asigna el tamaño de los buzones y Establece la cantidad de productos a producir o consumir por cada thread, y la cantidad de Productores y Consumidores, e Intermedios. Esta clase se compartirá con las clases: Productor, Intermedio y Consumidor.
* **Producto –** Esta clase determina que tipo de producto se está manipulando. Pueden ser de Tipo A oTipo B; con TipoA como un atributo boolean modificado por parámetro al momento de su creación por parte de la clase Productor (TipoA=True) /(TipoB=False).
* **Productor –** Esta clase cuenta con los atributos: Max\_Productos el cual determina el número máximo de productos; tipoA el cual es un boolean que determina el tipo de producto a crear; un buzonProductores en el cual se depositan los productos y es compartido con el resto de productores creados; tamañoBuzon, el cual representa la cantidad máxima de productos que un buzonProductores puede tener.
* En esta clase inicialmente verifica que la cantidad máxima de productos sea la correcta y procede a crear esos productos. Verifica y añade los productos al buzonProductores solo si la cantidad de productos es menor que el tamaño máximo del mismo buzón, y se le notifica a un thread intermedio que solo existe un producto en el buzón para que pueda ser manipulado.
* **Sincronización –** Para lograr la sincronización de Productores e intermedios, se implementaron bloques Syncronized adjuntos al buzonProductores para lograr un acceso ordenado a su utilización por parte de otros threads tanto para el retiro como el añadido de productos.
* El método yield () se invoca al momento de que el buzón se encuentre lleno, se realiza una espera semiactiva para que se entre al estado de listo y se retire al menos uno de los productos en él.
* También se emplea sincronización sobre el buzonProductores para la ejecución controlada del fragmento de código que ejecuta tanto añadir un producto creado anteriormente como el notifyall () que notifica a un intermedio que existe un producto en el buzón para transportarlo.

Se hacen impresiones en consola del estado del buzonProductores.

* **Intermedio –** Esta clase cuenta con un atributo boolean Izquierda, el cual indica si el intermediario es el de la izquierda o el de la derecha. También se implementa un atributo conteo, el cual representa el numero total de productos que se van a transportar en la ejecución del sistema. En la clase Intermedio se manipulan tres buzones: buzonProductores, buzonIntermedio, buzonConsumidores.

El intermediario Izquierda se encarga del manejo y acceso controlado al buzonProductores y el buzonIntermedio, mientras el Intermediario Derecha se encarga del buzonIntermedio y del manejo y acceso controlado al buzonConsumidores. Para la creación de un Intermedio se reciben tanto el boolean Izquierda, como los tres buzones y se inicializan con los atributos.

El funcionamiento está determinado por el tipo de intermedio, es decir, si se está ejecutando un Intermedio Izquierda o un Derecha.

El Intermediario Izquierda, mientras existan productos (señalados en conteo) para transportar, comienza verificando si existen productos en buzonProductores; en caso de que se encuentre vacío se realiza una espera activa. Cuando existen productos, retira un producto del buzonProductores y se actualiza una variable local para su manejo. Se verifica que mientras el buzonIntermedio cuente con un máximo de un producto este entre en estado de *espera activa*. Se añade el producto eliminado del buzonProductores al buzonIntermedio. Se notifica al otro thread Intermedio para que pueda remover productos del buzón y se reduce en una unidad la cantidad de productos totales.

El intermediario Derecha, mientras existan productos para transportar, verifica inicialmente si existen productos en el buzonIntermedio, si no se encuentra el único que debería existir dentro del buzón, se entra en estado de *espera activa*. Seguidamente se remueve un producto del buzonIntermedio, se notifica al segundo thread intermedio y se actualiza la variable de producto eliminado. Seguidamente se verifica que siempre y cuando el buzón se encuentre lleno, se entre en estado de espera activa hasta que haya un espacio, mientras se añade un producto eliminado al buzonConsumidor y se reduce la cantidad de productos totales que se transportan.

* **Sincronización –** Para la sincronización tanto de *Productor - Intermedio*, *Intermedio – Intermedio* e *Intermedio – Consumidor*, se implementaron bloques Syncronized adjuntos a la manipulación de los buzones. El acceso, añadido y retiro de productos de buzonProductores, buzonIntermedio y buzonConsumidores se sincroniza y se garantiza que solo un thread intermedio puede ejecutar operaciones entre el bloque de código y los buzones.
* El método *wait()* se invoca en los siguientes casos: Que el buzonProductores se encuentre vacío, por lo que se tendrá que hacer *espera activa* hasta que se encuentre al menos un producto; en el caso de que el buzonIntermedio esté lleno con su máximo que es un solo producto, se requiera añadir o que el mismo buzón se encuentre vacío y otro intermedio requiera retirar un producto, y como último caso de utilización cuando el buzonConsumidor se encuentre lleno y el Intermedio2 requiera añadir un producto.

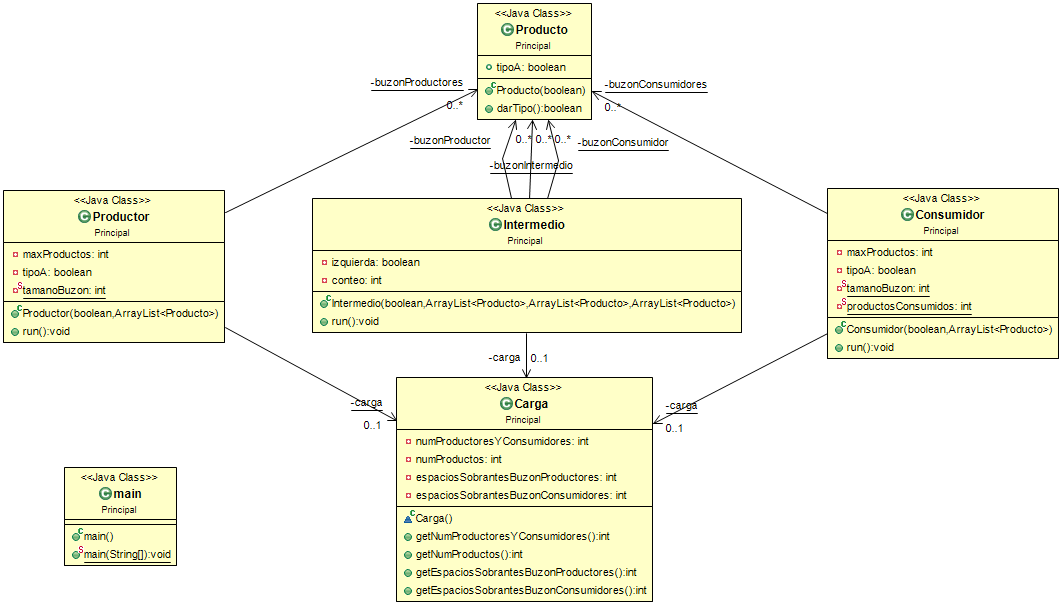
Se hacen impresiones en consola del tamaño de cada buzón después de su manipulación.

* **Consumidor –** En esta clase se implementaron los atributos: maxProductos, tipoA, buzonConsumidores, tamañoBuzon y productosConsumidos. El atributo productosConsumidos cuenta el número productos han sido consumidos y todos lo consumidores comparten esta variable. El resto de los atributos han sido señalados en las implementaciones de las anteriores clases.
* En esta clase inicialmente se verifica que todo se ejecute mientras existan productos, seguidamente se considera la sincronización para el manejo del buzón.
* La operación más importante de esta clase consiste en verificar mediante un for si el producto es del tipo que el Consumidor puede consumir (Tipo A / Tipo B). Si se encuentra un producto el cual se pueda consumir, se procede a remover el producto del buzonConsumidores y reducir la cantidad de productos, mientras se aumenta la variable compartida de productos consumidos.
* **Sincronización –** Para la sincronización de Intermedio – Consumidor se implementaron bloques de código Syncronized adjuntos al buzonConsumidor. Con este monitor se garantiza el acceso y eliminación de productos del buzonConsumidores mientras se detalla al mismo tiempo el tipo de Consumidor.
* El método *yield ()* se invoca al momento que no existan productos para consumir; se realiza una espera semiactiva hasta que se encuentre alguno.

Al instante de que se requiera consumir un producto, Si el buzón se encontraba lleno y se consume un producto, se notifica a un thread Intermedio que puede añadir productos al buzón.

En la clase se detallan los estados en que es consumido un producto y el tamaño del buzón por medio de impresiones en consola.

* **UML – CASO 1**

****