Miguel Ángel Ramírez Pérez mramper0604@g.educaand.es

Explicación - Ejercicio 04 - Logística de almacén

Este ejercicio simula un sistema de logística de un almacén donde hay vehículos de carga y descarga que se intercambian contenedores utilizando un Exchanger. Los vehículos de descarga llevan contenedores llenos y los intercambios por contenedores vacíos. Los vehículos de carga llevan contenedores vacíos y los intercambian por contenedores llenos.

El ejercicio está compuesto por las siguientes clases:

- **Enum Producto:** Define los tipos de productos que se pueden transportar.
- **Clase Contenedor:** Representa un contenedor con un ID y un contenido.
- **Clase VehiculoDescarga:** Representa vehículos que llevan contenedores llenos.
- **Clase VehiculoCarga:** Representa vehículos que llevan contenedores vacíos.
- **Clase Main:** Contiene toda la simulación.

Producto.java

Es un enum que define lo siguiente:

- NINGUNO: Indica que el contenedor está vacío.
- SERRIN: Producto de tipo serrín.
- NARANJAS: Producto de tipo naranjas.
- CANDADOS: Producto de tipo candados.
- AGUA: Producto de tipo agua.
- MADERA: Producto de tipo madera.

Este enum permite representar de una forma más clara y segura qué tipo de mercancía contiene cada contenedor.

Contenedor.java

Representa un contenedor de mercancías con las siguientes características:

Variables:

- **final int ID:** Identificador único del contenedor.
- **Producto contenido:** Producto actual que contiene el contenedor.

Métodos:

Constructor Contenedor(int id, Producto contenidoIncial):

- Inicializa el contenedor con un ID y un contenido inicial.
- El contenido puede ser NINGUNO si el contenedor comienza vacío.

llenar(Producto producto):

- Asigna un producto al contenedor.
- Cambia el estado del contenedor de vacío a lleno.

vaciar():

- Establece el contenido a Producto.NINGUNO.
- Cambia el estado del contenedor de lleno a vacío.

estaVacio():

- Devuelve true si el contenedor no contiene ningún producto (contenido == NINGUNO).
- Devuelve false si tiene algún producto.

toString():

- Devuelve el siguiente texto: ***"Contenedor <ID> en estos momentos contiene <producto>"***.
- Muestra el estado del contenedor en la consola.

VehiculoDescarga.java

Esta clase implementa ***Runnable*** y representa un vehículo que llega al almacén con contenedores llenos, los intercambia por contenedores vacíos, y luego llena esos contenedores vacíos con nuevos productos.

Variables:

- **puntoIntercambio:** Exchanger compartido donde se realiza el intercambio de contenedores.
- **miContenedor:** Contenedor actual que posee este vehículo.
- **numCiclos:** Número de ciclos que realizará (entre 3 y 5).

Constructor:
Recibe tres parámetros:

- El Exchanger compartido por todos los vehículos.
- Un contenedor inicial (puede estar vacío al inicio).
- El número de ciclos.

Método run():
Define el comportamiento del hilo. En cada ciclo realiza:

1. **Llena el contenedor:** Selecciona aleatoriamente un producto del array *{SERRIN, NARANJAS, CANDADOS, AGUA, MADERA}* y llena el contenedor usando el método ***llenar()***.
2. **Anuncia la preparación:** *"Vehículo de descarga. Ciclo X Contenedor preparado para INTERCAMBIAR: <contenedor>".*
3. **Anuncia la espera:** *"Vehículo de descarga: esperando punto de encuentro para intercambiar por un contenedor vacío..."*
4. **Intercambia el contenedor:** Llama a ***puntoIntercambio.exchange(miContenedor)***. Esta línea es crítica:
    - El hilo se bloquea aquí hasta que el VehiculoCarga también llegue al intercambio.
    - Cuando ambos vehículos están listos, se produce el intercambio.
    - El VehiculoDescarga entrega su contenedor lleno y recibe uno vacío.
5. **Muestra el intercambio**: *"Vehículo de descarga: se ha hecho el intercambio. Contenedor recibido: <contenedor>"*.

6. Verifica el contenedor recibido:
    - Si ***miContenedor.estaVacio()*** es true: *"contenedor vacío. Listo para el próximo ciclo."*
    - Si es false. *"ERROR! El contenedor recibido no estaba vacío."*
7. **Espera entre ciclos:** Si no es el último ciclo, espera 3 segundos con ***Thread.sleep(3000)*** antes de la siguiente iteración.
8. **Mensaje final:** Al terminar todos los ciclos: *"Vehículo de descarga: trabajo finalizado!"*

Manejo de excepciones:
Todo este código está dentro de un bloque try-catch. Si el hilo es interrumpido durante ***exchange()*** o ***sleep()***, captura la ***InterruptedException*** y marca el hilo como interrumpido con ***Thread.currentThread().interrupt()***.

VehiculoCarga.java

Esta clase también implementa ***Runnable*** y representa un vehículo que llega al almacén con contenedores vacíos, los intercambia por contenedores llenos, y luego vacía esos contenedores.

Variables:

- **puntoIntercambio:** Exchanger compartido (el mismo que usa VehiculoDescarga).
- **miContenedor:** Contenedor actual del vehículo.
- **numCiclos:** Número de ciclos a realizar (el mismo número que VehiculoDescarga).

Constructor:
Idéntico al de VehiculoDescarga: recibe el Exchanger, el contenedor inicial y el número de ciclos.

Método run():
En cada ciclo realiza:

1. **Vacía el contenedor:** Llama a ***miContenedor.vaciar()*** para asegurarse de que el contenedor está vacío antes del intercambio.
2. **Anuncia que está preparado:** *"Vehículo de carga. Ciclo X Contenedor preparado para INTERCAMBIAR: <contenedor>".*
3. **Anuncia la espera:** *"Vehículo de carga: esperando punto de encuentro para intercambiar por un contenedor lleno..."*
4. **Intercambia contenedor:** Llama a ***puntoIntercambio.exchange(miContenedor)***:
    - Se bloquea hasta que VehiculoDescarga también llegue al punto.
    - Entrega su contenedor vacío y recibe uno lleno.

5. **Muestra el intercambio:** Muestra: *"Vehículo de carga: se ha hecho el intercambio. Contenedor recibido: <contenedor>".*
6. **Verifica el contenedor recibido:**
    - Si ***!miContenedor.estaVacio()*** es true (contenedor lleno): *"contenedor lleno. Vaciando contenedor..."*, luego llama a ***vaciar()***, y finalmente *"contenedor vaciado. Listo para el próximo ciclo."*
    - Si está vacío: *"ERROR! El contenedor recibido estaba vacío."*

7. **Espera entre ciclos:** Si no es el último ciclo, espera 3 segundos.
8. **Mensaje final:** Al terminar: *"Vehículo de carga: trabajo finalizado!"*

Manejo de excepciones:
Todo este código está dentro de un bloque try-catch. Si el hilo es interrumpido durante ***exchange()*** o ***sleep()***, captura la ***InterruptedException*** y marca el hilo como interrumpido con ***Thread.currentThread().interrupt()***.

Main.java

Contiene el método principal que ejecuta la simulación:

Proceso de ejecución:

Crea el punto de intercambio: Instancia un objeto ***Exchanger<Contenedor>*** que será compartido por ambos vehículos. Este objeto es esencial para la sincronización del intercambio.

Crea los contenedores iniciales: Crea dos contenedores, ambos inicialmente vacíos (Producto.NINGUNO):

- Contenedor ID 1 para VehiculoDescarga
- Contenedor ID 2 para VehiculoCarga

Genera número de ciclos aleatorio: Usa ***Random*** para generar un número entre 3 y 5 mediante la expresión ***3 + random.nextInt(3)***.

Mensaje inicial:
--- INICIO SIMULACIÓN LOGÍSTICA DE CONTENEDORES ---
Vehículo de descarga comienza con: <contenedor 1>
Vehículo de carga comienza con: <contenedor 2>

Crea los vehículos:

- VehiculoDescarga con el Exchanger, contenedor 1 y número de ciclos
- VehiculoCarga con el Exchanger, contenedor 2 y el mismo número de ciclos

Crea e inicia los hilos:
Crea dos objetos Thread con nombres descriptivos ("Vehículo de descarga" y "Vehículo de carga"). Llama a ***start()*** en ambos hilos. Cada vehículo comienza a ejecutar su método ***run()*** de forma concurrente.

Espera la finalización:
Usa ***join()*** en cada hilo. Esto hace que el hilo main se bloquee esperando a que cada vehículo termine todos sus ciclos. Solo cuando ambos ***join()*** finalizan, main continúa.

Mensaje final: Todos los vehículos han completado sus ciclos. Fin del programa.