Repaso del Tema 2.
Bloqueo con estados y productor-consumidor

Enunciado de la práctica:

Se desea simular un taller mecánico **"Talleres Santi"**, en Java donde existan 5 mecánicos que trabajan de forma concurrente y comparten un recurso limitado: Un carrito con **3 juegos de destornilladores**.

Características del taller:

1. Cada mecánico es un hilo independiente que representa a un mecánico trabajando en el taller.
2. Cuando llega un coche, un mecánico necesita un juego de destornilladores para trabajar.
3. Si no hay destornilladores disponibles, el mecánico debe esperar hasta que alguno se libere.
4. El mecánico usa el destornillador durante un tiempo aleatorio entre un mínimo y un máximo (por ejemplo, 300-800 ms).
5. Después de usarlo, libera el destornillador para que otros puedan usarlo.
6. Cada mecánico procesa un número limitado de coches (por ejemplo, 8) o indefinidamente.
7. La sincronización debe hacerse con **`synchronized(this)`** dentro del recurso compartido.
8. El programa debe mostrar por consola el estado de cada mecánico y del carrito de destornilladores en cada acción.

Objetivos:

- Practicar la sincronización de hilos en Java.
- Evitar interbloqueos al manejar recursos compartidos.
- Usar **`wait()`** y **`notifyAll()`** para la coordinación de hilos.

<u>Solución sencilla:</u>

```
class CarritoDestornilladores {
    private int juegosDisponibles; //Para que podáis cambiarlo.

    public CarritoDestornilladores(int juegos) {
        juegosDisponibles = juegos;
    }

    public void adquirir(String nombre) throws InterruptedException {
        synchronized (this) {
            while (juegosDisponibles == 0) {
                System.out.printf("[%s] -> ESPERANDO DESTORNILLADOR
            (disponibles=%d)%n", nombre, juegosDisponibles);
```

```
                        this.wait();
                    }//fin while

                    juegosDisponibles--;
                    System.out.printf("[%s] -> USANDO DESTORNILLADOR
            (disponibles=%d)%n", nombre, juegosDisponibles);
            } //fin synchronized
        }//fin adquirir


        public void liberar(String nombre) {
            synchronized (this) {
                juegosDisponibles++;
                System.out.printf("[%s] ha liberado el destornillador
            (disponibles=%d)%n", nombre, juegosDisponibles);
                this.notifyAll();
            }
        }

        public synchronized String estado() {
            return String.format("Carrito(destornilladores=%d)", juegosDisponibles);
        }
}




class Mecanico extends Thread {
        private final CarritoDestornilladores carrito;
        private final Random rnd = new Random();
        private final int minEntreCoches;
        private final int maxEntreCoches;
        private final int minUso;
        private final int maxUso;
        private final int iteraciones;

        public Mecanico(CarritoDestornilladores carrito, String nombre,
            int minEntreCoches, int maxEntreCoches,
            int minUso, int maxUso, int iteraciones)
        {
            super(nombre);//sobreescribimos el name del Thread (clase padre)
            this.carrito = carrito;
            this.minEntreCoches = minEntreCoches;
            this.maxEntreCoches = maxEntreCoches;
            this.minUso = minUso;
            this.maxUso = maxUso;
            this.iteraciones = iteraciones;
        }
```

```
	private int aleatorio(int min, int max) {
		return min + rnd.nextInt(max - min + 1);
	}

	private void imprimirEstado(String estado) {
		System.out.printf("[%s] -> %s%n", getName(), estado);
	}


	@Override
	public void run() {
		int procesados = 0;
		try {
			//si iteraciones es negativo, queremos que trabaje siempre.
			while (iteraciones < 0 || procesados < iteraciones) {
				imprimirEstado("INACTIVO");
				Thread.sleep(aleatorio(minEntreCoches, maxEntreCoches));

				System.out.printf("[%s] ha llegado un coche. (%s)%n",
			getName(), carrito.estado());

				carrito.adquirir(getName());

				int uso = aleatorio(minUso, maxUso);
				System.out.printf("[%s] usando destornillador durante %d
			ms%n", getName(), uso);
				Thread.sleep(uso);

				carrito.liberar(getName());

				imprimirEstado("TRABAJANDO");
				Thread.sleep(aleatorio(50, 150));

				procesados++;
			}
		} catch (InterruptedException e) {
			System.out.println("Hilo interrumpido, saliendo...");
			 System.exit(0);// termina todo el programa. Finalizan todos los hilos.
		} finally {
			imprimirEstado("TERMINADO");
			System.out.printf("[%s] -> TERMINADO (coches
		procesados=%d)%n", getName(), procesados);
		}
	}
} //fin clase
```

```
import java.util.Random;

public class TallerMecanicoDestornilladores {

	public static void main(String[] args) throws InterruptedException {
		CarritoDestornilladores carrito = new CarritoDestornilladores(3);

		final int NUM_MECANICOS = 5;

		//Si pasamos iteraciones = -1, haremos bucle infinito en el hilo.
		final int ITERACIONES = 8;  //Atenderá a 8 coches cada mecánico

		final int MIN_ENTRE_COCHES = 200;
		final int MAX_ENTRE_COCHES = 700;
		final int MIN_USO = 300;
		final int MAX_USO = 800;

		Mecanico[] taller = new Mecanico[NUM_MECANICOS];
		for (int i = 0; i < NUM_MECANICOS; i++) {
			String nombre = "Mecanico-" + (i + 1);
			taller[i] = new Mecanico(carrito, nombre, MIN_ENTRE_COCHES,
			MAX_ENTRE_COCHES, MIN_USO, MAX_USO, ITERACIONES);
			taller[i].start();
		}

		for (Mecanico m : taller)
			m.join();

		System.out.println("=== Taller: todos los mecánicos han terminado ===");
	}
}
```

Se pide:

1. Probar, documentar y entender este ejemplo. Nombrar a cada mecánico.
2. Permite que pasemos el número de destornilladores como args.
3. Realizar ahora una pequeña implementación, basada en que existe un hilo llamado **GeneradorCoches**, que cada cierto tiempo, genera un coche y notifica a los mecánicos. (Deberéis ramificar, porque vais a generar una nueva implementación)
    a. En este ejemplo, quiero que os creéis un recurso compartido llamado Taller, con un atributo **cochesPendientes**, que vaya contando los coches que hay por atender. Deberéis crear dos métodos **tomarCoche** y **llegadaCoche**. Llegada coche, hará una notificación a todos, mientras que **tomarCoche**, deberá bloquearse si no hay **cochesPendientes**.
    b. Suponer que el generador de coches, acepta un coche entre **200** y **500 msg**.
    c. El mecánico, deberá en un bucle **while (true)**, tomar coches y dormirse **300 msg**, hasta intentar tomar otro coche.
4. Mezclar los dos ejemplos, el original con el recurso compartido de **CarritoDestornilladores** y el **punto 3**, en el que tendréis otro recurso compartido. De modo que:
    a. Deberéis quitar el while (iteraciones ….) y poner un while(true), porque no queremos que acabe nunca.
    b. El mecánico deberá solicitar tomar un coche.
    c. Una vez tomado el coche, deberá solicitar las herramientas.
    d. Se deberá dormir 300 msg.
    e. Algo como:

```
public void run() {
        try {
                while (true) {
                        taller.tomarCoche(getName());
                        //Pedimos destornillador. Punto 1.
                        Thread.sleep(300); //Lo que tarda.
        }
        } catch (InterruptedException e) {
                System.out.println(getName() + " interrumpido");
        }
}
```

5. Imaginar ahora, que hubiera un máximo de coches a generar. Queremos que los mecánicos finalicen, al comprobar alguna variable que deba poner el generador de coches, como que ya no llegarán más. Deberán acabar su jornada laboral y todos los hilos deberán informar del número de coches atendidos.

6. Crear otro hilo de ejecución llamado **Gerente**, tal que esté dormido mucho tiempo esperando una interrupción por medio del generador de coches. En el momento que finalicen la jornada laboral, se deberá interrumpir al Gerente y éste hará que cierre el taller.