Tomás Pérez Carrillo
2º DAM

Actividad 02. Programación Multihilo

1. Crear un programa con tres hilos que escriban en un mismo archivo (reciben el nombre por parámetro y habrá que crearlo si no existe). El primero de ellos escribirá los números del 11 al 20; el segundo del 21 al 30; y el tercero del 1 al 10. Tras probar su funcionamiento, intenta conseguir que se graben en orden (aunque se pierda el efecto multihilo).

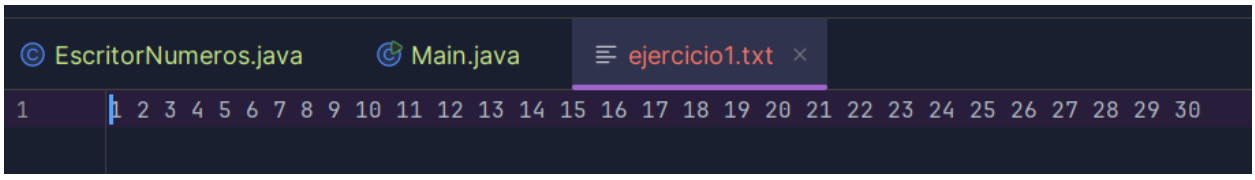
Código:

```
public class EscritorNumeros implements Runnable {  6 usages  new *

    private String nombreFichero;  2 usages
    private int inicio;  2 usages
    private int fin;  2 usages

    public EscritorNumeros(String nombreFichero, int inicio, int fin) {  3 usages
        this.nombreFichero = nombreFichero;
        this.inicio = inicio;
        this.fin = fin;
    }

    @Override  new *
    public void run() {

        try (FileWriter fw = new FileWriter(nombreFichero,  append: true);
             BufferedWriter bw = new BufferedWriter(fw)) {

            for (int i = inicio; i <= fin; i++) {
                bw.write( str: i+" ");

                Thread.sleep( millis: 100);
            }

        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }

    }
}
```

Tomás Pérez Carrillo
2º DAM

```java
public class Main {  new *

    private static final String nombreArchivo = "src/main/java/Ejercicio1/ejercicio1.txt";

    public static void main(String[] args) {  new *


        EscritorNumeros escritor1 = new EscritorNumeros(nombreArchivo,  inicio: 11,  fin: 20);
        Thread e1 = new Thread(escritor1);

        EscritorNumeros escritor2 = new EscritorNumeros(nombreArchivo,  inicio: 21,  fin: 30);
        Thread e2 = new Thread(escritor2);

        EscritorNumeros escritor3 = new EscritorNumeros(nombreArchivo,  inicio: 1,  fin: 10);
        Thread e3 = new Thread(escritor3);

        try {
            e3.start();
            e3.join();

            e1.start();
            e1.join();

            e2.start();
            e2.join();
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

    }

}
```

Salida en fichero:



Comentarios:

Para este ejercicio he utilizado try with resources ya que lo vimos hace poco en acceso a datos y viene muy bien para el tema de cerrar los recursos automáticamente y garantizar que no haya error en ese aspecto. Por lo demás, se han creado los tres hilos heredando de runnable y se han sincronizado usando el método join para ejecutarlos por orden y obtener el resultado esperado en el archivo “ejercicio1.txt”.

Tomás Pérez Carrillo
2º DAM

2. Programa una aplicación que muestre la hora en dos lugares distintos del mundo por terminal. Debemos programar estos usando hilos. Esta hora se deberá actualizar cada cinco segundos y deberá empezar a mostrarse tres segundos después del momento en el que se arranque la aplicación. Tras 22 segundos se dejará de mostrar la hora.

```java
public class Relojes implements Runnable {  2 usages  new *

    @Override  new *
    public void run() {
        // Formato para la hora
        DateTimeFormatter formato = DateTimeFormatter.ofPattern("HH:mm:ss");

        LocalTime horaMadrid = LocalTime.now();
        LocalTime horaNY = LocalTime.now(ZoneId.of( zoneId: "America/New_York"));


        System.out.println("Madrid: " + horaMadrid.format(formato) +
                " | Nueva York: " + horaNY.format(formato));
    }
}
```

```java
public class Main {  new *

    public static void main(String[] args) {  new *

        ScheduledExecutorService planificador = Executors.newSingleThreadScheduledExecutor();

        Relojes miReloj = new Relojes();

        System.out.println("Iniciando aplicación... (El reloj comenzará en 3 segundos)");

        planificador.scheduleAtFixedRate(miReloj, initialDelay: 3, period: 5, TimeUnit.SECONDS);

        //Lambda para apagar el planificador a los 22 segundos
        planificador.schedule(() -> {
            System.out.println("Tiempo cumplido (22s). Deteniendo el reloj.");
            planificador.shutdown();
        }, delay: 22, TimeUnit.SECONDS);
    }
}
```

Tomás Pérez Carrillo
2º DAM

Salida por terminal:

```
> Task :Ejercicio2.Main.main()
Iniciando aplicaci�n... (El reloj comenzar� en 3 segundos)
Madrid: 12:23:47 | Nueva York: 06:23:47
Madrid: 12:23:52 | Nueva York: 06:23:52
Madrid: 12:23:57 | Nueva York: 06:23:57
Madrid: 12:24:02 | Nueva York: 06:24:02
Tiempo cumplido (22s). Deteniendo el reloj.

BUILD SUCCESSFUL in 22s
```

Comentarios:

Para resolver este ejercicio se ha utilizado “ScheduledExecutorService” ya que ofrece un control de los hilos y manejo de tiempos bastante intuitivo pudiendo planificar el delay inicial (en este caso 3s) y cada cuanto se va a ejecutar el hilo (period). Además, he utilizado una lambda para cerrar el planificador ya que en Android las frecuentamos mucho y vienen muy bien para escribir menos código, aunque la sintaxis a veces pueda ser un poco liosa. No obstante, intentaré usarlas más a menudo ya que más o menos les estoy pillando el truco.

Tomás Pérez Carrillo
2º DAM

3. Implementa un sistema productor–consumidor utilizando hilos donde un productor genere cadenas de 15 caracteres y las vaya almacenando en un búfer compartido con capacidad máxima de 6 caracteres. Paralelamente, un consumidor deberá ir extrayendo caracteres de ese mismo búfer hasta completar también 15 caracteres. El consumidor solo podrá leer cuando el búfer no esté vacío, y el productor únicamente podrá escribir cuando quede espacio disponible. Ambos hilos deben coordinarse correctamente para evitar condiciones de carrera y garantizar que la producción y el consumo se realizan de forma sincronizada. Muestra en consola las acciones de producción, consumo y estado del búfer. Introduce pausas tras cada letra generada y letra consumida (el tiempo será menor al generar que al consumir).

Código:

```java
import java.util.LinkedList;

public class Buffer {  6 usages  new *

    private LinkedList<Character> lista;  5 usages

    private int capacidad;  2 usages

    public Buffer(int capacidad1) {  1 usage  new *
        lista = new LinkedList<>();
        this.capacidad = capacidad1;
    }

    public synchronized void producirLetra(char letra) throws InterruptedException {  1 usage

        //Si la lista del buffer está llena:
        while (lista.size() == capacidad) {
            System.out.println("Bufer lleno. Productor espera");
            wait();
        }

        lista.add(letra);
        System.out.println("Letra agregada al bufer: " + letra);
        notifyAll();
    }

    public synchronized void consumirLetra() throws InterruptedException {  1 usage  new *

        while (lista.isEmpty()) {
            System.out.println("Búfer sin letras para consumir. Consumidor espera.");
            wait();
        }

        char letra = lista.removeFirst();
        System.out.println("Letra consumida " + letra);
        notifyAll();
    }
```

Tomás Pérez Carrillo
2º DAM

```java
public class Productor implements Runnable {  2 usages   new
    private Buffer buffer;  2 usages
    public Productor(Buffer buffer) {  1 usage   new *
        this.buffer = buffer;
    }
    @Override  new *
    public void run() {
        try {
            for (int i = 0; i < 15 ; i++) {
                char letra = (char) ('A' + i);
                buffer.producirLetra(letra);
                Thread.sleep( millis: 350);//pausa breve
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
public class Consumidor implements Runnable{  2 usages
    private Buffer buffer;  2 usages
    public Consumidor(Buffer buffer) {  1 usage   new *
        this.buffer = buffer;
    }
    @Override  new *
    public void run() {
        try{
            for (int i = 0; i < 15; i++) {
                buffer.consumirLetra();
                Thread.sleep( millis: 1000);
            }
        }catch(InterruptedException e){
            e.printStackTrace();
        }
    }
}
```

Tomás Pérez Carrillo
2º DAM

Salida por terminal:

```
> Task :Ejercicio3.Main.main()
Letra agregada al bufer: A
Letra consumida A
Letra agregada al bufer: B
Letra agregada al bufer: C
Letra consumida B
Letra agregada al bufer: D
Letra agregada al bufer: E
Letra agregada al bufer: F
Letra consumida C
Letra agregada al bufer: G
Letra agregada al bufer: H
Letra agregada al bufer: I
Letra consumida D
Letra agregada al bufer: J
Bufer lleno. Productor espera
Letra consumida E
Letra agregada al bufer: K
Bufer lleno. Productor espera
Letra consumida F
Letra agregada al bufer: L
Bufer lleno. Productor espera
Letra consumida G
Letra agregada al bufer: M
Bufer lleno. Productor espera
Letra consumida H
Letra agregada al bufer: N
Bufer lleno. Productor espera
Letra consumida I
Letra agregada al bufer: O
Letra consumida J
Letra consumida K
Letra consumida L
Letra consumida M
Letra consumida N
Letra consumida O

BUILD SUCCESSFUL in 15s
2 actionable tasks: 2 executed
```

Tomás Pérez Carrillo
2º DAM

Comentarios:

Para la implementación del búfer compartido he optado por una LinkedList en lugar del habitual ArrayList. Aunque suelo usar este último por inercia, en este caso la lista enlazada es mucho más eficiente para una cola FIFO, ya que extraer el primer elemento (removeFirst) no obliga a desplazar el resto de datos en memoria. Me ha parecido un detalle importante para optimizar el rendimiento del consumidor, además de reforzar el uso de wait() y notifyAll() para la coordinación.

4. Usando semáforos. Cinco filósofos pasan su vida pensando y comiendo. Los filósofos comparten una mesa circular rodeada por cinco sillas, una para cada uno de ellos. En el centro de la mesa se encuentra una fuente de arroz, y también sobre la mesa hay cinco palillos chinos. Cuando un filósofo piensa, no interactúa con sus colegas. Ocasionalmente, un filósofo tiene hambre y trata de coger los dos palillos que están más cerca de él (los palillos colocados entre él y sus vecinos de la derecha y de la izquierda). Un filósofo sólo puede coger un palillo a la vez y, obviamente, no puede ser el que está en la mano de un vecino. Cuando un filósofo hambriento tiene sus dos palillos al mismo tiempo, come sin soltarlos. Cuando termina de comer, coloca ambos palillos sobre la mesa y comienza a pensar otra vez. Se valorará positivamente hacer este ejercicio con número variable de filósofos (se puede preguntar al principio y recoger ese dato).

```java
public class Filosofo implements Runnable {  2 usages  new *

    private int id;  5 usages
    private Semaphore palilloIzquierdo;  4 usages
    private Semaphore palilloDerecho;  4 usages

    public Filosofo(int id, Semaphore palilloIzquierdo, Semaphore palilloDerecho) {  1 usage
        this.id = id;
        this.palilloIzquierdo = palilloIzquierdo;
        this.palilloDerecho = palilloDerecho;
    }

    @Override  new *
    public void run() {
        try {
            while (true) { // Bucle infinito: pensar -> comer -> pensar

                System.out.println("Filósofo " + id + " pensando...");
                Thread.sleep((long) (Math.random() * 2000)); // Tiempo aleatorio pensando

                if (id % 2 == 0) {
                    // Filósofos PARES: Izquierda primero, luego Derecha
                    palilloIzquierdo.acquire();
                    palilloDerecho.acquire();
                } else {
                    // Filósofos IMPARES: Derecha primero, luego Izquierda
                    palilloDerecho.acquire();
                    palilloIzquierdo.acquire();
                }

                System.out.println("Filósofo " + id + " se dispone a comer.");

                // Tiempo comiendo
                Thread.sleep((long) (Math.random() * 2000));

                palilloDerecho.release();
                palilloIzquierdo.release();

                System.out.println("Filósofo " + id + " terminó y soltó los palillos.");
            }
        } catch (InterruptedException e) {
            return;
        }
```

Tomás Pérez Carrillo
2º DAM

```java
public class Mesa {  new *

    public static void main(String[] args) {  new *

        Semaphore[] palillos = new Semaphore[5];

        for (int i = 0; i < 5; i++) {
            palillos[i] = new Semaphore( permits: 1);
        }

        for (int i = 0; i < 5; i++) {
            Semaphore palilloIzq = palillos[i];

            //palillo derecho del ultimo = palillo izquierdo 1er filosofo
            Semaphore palilloDer = palillos[(i + 1) % 5];

            Filosofo f = new Filosofo(i, palilloIzq, palilloDer);
            new Thread(f).start();
        }

        //Para que no dure infinitamente el programa
        try {
            Thread.sleep( millis: 15000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Se acabó la cena. Cerrando programa.");
        System.exit( status: 0);
    }

}
```

Tomás Pérez Carrillo
2º DAM

Salida por terminal:

```
> Task :Ejercicio4.Mesa.main()
Fil◆sofo 3 pensando...
Fil◆sofo 1 pensando...
Fil◆sofo 2 pensando...
Fil◆sofo 4 pensando...
Fil◆sofo 0 pensando...
Fil◆sofo 1 se dispone a comer.
Fil◆sofo 3 se dispone a comer.
Fil◆sofo 1 termin◆ y solt◆ los palillos.
Fil◆sofo 1 pensando...
Fil◆sofo 0 se dispone a comer.
Fil◆sofo 3 termin◆ y solt◆ los palillos.
Fil◆sofo 3 pensando...
Fil◆sofo 2 se dispone a comer.
Fil◆sofo 0 termin◆ y solt◆ los palillos.
Fil◆sofo 0 pensando...
Fil◆sofo 4 se dispone a comer.
Fil◆sofo 2 termin◆ y solt◆ los palillos.
Fil◆sofo 2 pensando...
Fil◆sofo 4 termin◆ y solt◆ los palillos.
Fil◆sofo 4 pensando...
Fil◆sofo 1 se dispone a comer.
Fil◆sofo 1 termin◆ y solt◆ los palillos.
Fil◆sofo 1 pensando...
Fil◆sofo 3 se dispone a comer.
Fil◆sofo 0 se dispone a comer.
Fil◆sofo 3 termin◆ y solt◆ los palillos.
Fil◆sofo 3 pensando...
Fil◆sofo 2 se dispone a comer.
Fil◆sofo 0 termin◆ y solt◆ los palillos.
Fil◆sofo 0 pensando...
```

Tomás Pérez Carrillo
2º DAM

Comentarios:

Este ejercicio me ha costado bastante el tema de asignar los palillos y he tenido que buscar esta parte:

```
//palillo derecho del ultimo = palillo izquierdo 1er filosofo
Semaphore palilloDer = palillos[(i + 1) % 5];
```

Esto lo que hace, cómo se puede leer en el comentario, es recrear la mesa redonda y que no sea una línea recta. También he filtrado a los filósofos por su id, los pares empiezan a coger los palillos de izquierda a derecha y los impares al revés.

Además, para evitar el bucle infinito he añadido en el main que se cierre el programa a los 15 segundos. Me gustaría que este ejercicio lo resolvieras en clase para ver si así se puede entender un poco mejor ya que me he ayudado bastante de diversas fuentes para sacarlo.

Tomás Pérez Carrillo
2º DAM

5. Resuelve el mismo ejercicio (filósofos), pero con el uso de monitores (synchronized).

```java
public class Filosofo implements Runnable {  2 usages  new *

    private int id;  5 usages
    private Object palilloIzquierdo;  3 usages
    private Object palilloDerecho;  3 usages

    public Filosofo(int id, Object palilloIzquierdo, Object palilloDerecho) {  1 usage  new *
        this.id = id;
        this.palilloIzquierdo = palilloIzquierdo;
        this.palilloDerecho = palilloDerecho;
    }

    @Override  new *
    public void run() {
        try {
            while (true) {
                System.out.println("Filósofo " + id + " pensando.");
                Thread.sleep((long) (Math.random() * 1000));
                if (id % 2 == 0) {
                    synchronized (palilloIzquierdo) { // Coge el primero
                        synchronized (palilloDerecho) { // Coge el segundo
                            comer();
                        }
                    }
                } else {
                    synchronized (palilloDerecho) {
                        synchronized (palilloIzquierdo) {
                            comer();
                        }
                    }
                }
            }
        } catch (InterruptedException e) {
            return;
        }
    }

    private void comer() throws InterruptedException {  2 usages  new *
        System.out.println("Filósofo " + id + " se dispone a comer.");
        Thread.sleep((long) (Math.random() * 1000));
        System.out.println("Filósofo " + id + " terminó de comer y soltó los palillos.");
    }
}
```

Tomás Pérez Carrillo
2º DAM

```java
public class Main {  new *

    public static void main(String[] args) {  new *

        Object[] palillos = new Object[5];
        for (int i = 0; i < 5; i++) {
            palillos[i] = new Object();
        }

        for (int i = 0; i < 5; i++) {
            Object palilloIzq = palillos[i];
            Object palilloDer = palillos[(i + 1) % 5]; // Misma forma circular

            Filosofo f = new Filosofo(i, palilloIzq, palilloDer);
            new Thread(f).start();
        }

        try {
            Thread.sleep( millis: 15000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Se acabó la fiesta.");
        System.exit( status: 0);
    }
}
```

Tomás Pérez Carrillo
2º DAM

Salida por terminal:

```
> Task :Ejercicio5.Main.main()
Fil◆sofo 4 pensando.
Fil◆sofo 0 pensando.
Fil◆sofo 3 pensando.
Fil◆sofo 1 pensando.
Fil◆sofo 2 pensando.
Fil◆sofo 3 se dispone a comer.
Fil◆sofo 3 termin◆ de comer y solt◆ los palillos.
Fil◆sofo 3 pensando.
Fil◆sofo 2 se dispone a comer.
Fil◆sofo 0 se dispone a comer.
Fil◆sofo 2 termin◆ de comer y solt◆ los palillos.
Fil◆sofo 2 pensando.
Fil◆sofo 3 se dispone a comer.
Fil◆sofo 0 termin◆ de comer y solt◆ los palillos.
Fil◆sofo 0 pensando.
Fil◆sofo 1 se dispone a comer.
Fil◆sofo 3 termin◆ de comer y solt◆ los palillos.
Fil◆sofo 3 pensando.
Fil◆sofo 4 se dispone a comer.
Fil◆sofo 4 termin◆ de comer y solt◆ los palillos.
Fil◆sofo 4 pensando.
Fil◆sofo 4 se dispone a comer.
Fil◆sofo 1 termin◆ de comer y solt◆ los palillos.
Fil◆sofo 1 pensando.
Fil◆sofo 2 se dispone a comer.
Fil◆sofo 2 termin◆ de comer y solt◆ los palillos.
Fil◆sofo 2 pensando.
Fil◆sofo 1 se dispone a comer.
Fil◆sofo 4 termin◆ de comer y solt◆ los palillos.
Fil◆sofo 4 pensando.
Fil◆sofo 3 se dispone a comer.
Fil◆sofo 1 termin◆ de comer y solt◆ los palillos.
Fil◆sofo 1 pensando.
```

Tomás Pérez Carrillo
2º DAM

Comentarios:

En esta variante del problema de los filósofos, he sustituido los semáforos por el mecanismo intrínseco de Java: los monitores (synchronized). En este caso, los palillos se representan mediante instancias genéricas de la clase Object. La diferencia fundamental en la implementación es que, para poseer dos recursos simultáneamente, es necesario anidar los bloques synchronized: una vez dentro del bloque del primer palillo, se intenta acceder al bloque del segundo. Al igual que en el ejercicio anterior, he mantenido la estrategia de ruptura de simetría (pares/impares) para prevenir el bloqueo, ya que el riesgo de quedarse esperando eternamente en el segundo synchronized sigue existiendo si todos compiten en el mismo orden.

6. Archena tiene a su entrada al pueblo un Puente sobre el río Segura. El puente es estrecho y no permite el cruce de dos coches que circulen en sentidos contrarios. Una vez que un coche llega e intenta cruzar por su lado (existen dos lados, uno de entrada al pueblo y otro de salida) este podrá cruzar si no existen coches en sentido contrario (hay coches en el puente cruzando en su sentido o directamente no hay coches). Diseñe un programa concurrente, libre de bloqueos e inaniciones, que simule el sistema descrito, tratando que:
• Nunca crucen simultáneamente el puente vehículos en sentidos opuestos.
• Todos pasen lo más pronto posible cumpliendo lo indicado.

```
public class Puente {  4 usages  new *

    private int cochesEnPuente = 0;  6 usages
    private String sentidoActual = null; // "NORTE", "SUR" o null (vacío)  4 usages

    public synchronized void entrar(String sentido, String nombreCoche) throws InterruptedException {  1 usage  new *

        while (cochesEnPuente > 0 && !sentido.equals(sentidoActual)) {
            System.out.println(nombreCoche + " (" + sentido + ") debe ESPERAR. Puente ocupado por " + sentidoActual);
            wait();
        }

        sentidoActual = sentido;
        cochesEnPuente++;

        System.out.println(nombreCoche + " entra al puente dirección " + sentido +
                ". (Coches dentro: " + cochesEnPuente + ")");
    }

    public synchronized void salir(String nombreCoche) {  1 usage  new *
        cochesEnPuente--;
        System.out.println(nombreCoche + " sale del puente. (Quedan: " + cochesEnPuente + ")");

        if (cochesEnPuente == 0) {
            senti
            Syste                                              rar coches de cualquier lado.");
        }

        notifyAll
    }

}
```

Tomás Pérez Carrillo
2º DAM

```java
public class Coche implements Runnable {  2 usages  new *

    private String nombre;  4 usages
    private String sentido; // "NORTE" o "SUR"  3 usages
    private Puente puente;  3 usages

    public Coche(String nombre, String sentido, Puente puente) {  1 usage  new *
        this.nombre = nombre;
        this.sentido = sentido;
        this.puente = puente;
    }

    @Override  new *
    public void run() {
        try {

            System.out.println(nombre + " llega desde el " + sentido + ".");

            puente.entrar(sentido, nombre);

            Thread.sleep( millis: (long) (Math.random() * 1000) + 500);

            puente.salir(nombre);

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```java
public class Main {  new *

    public static void main(String[] args) {  new *

        Puente puenteArchena = new Puente();

        String[] sentidos = {"NORTE", "SUR"};

        for (int i = 1; i <= 10; i++) {
            //Sentido al azar
            String sentidoAsignado = sentidos[(int)(Math.random() * 2)];

            Coche c = new Coche( nombre: "Coche-" + i, sentidoAsignado, puenteArchena);
            new Thread(c).start();

            // Pequeña pausa entre llegadas para que no lleguen todos en el milisegundo 0
            try {
                Thread.sleep((long) (Math.random() * 500));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Tomás Pérez Carrillo
2º DAM

Salida por terminal:

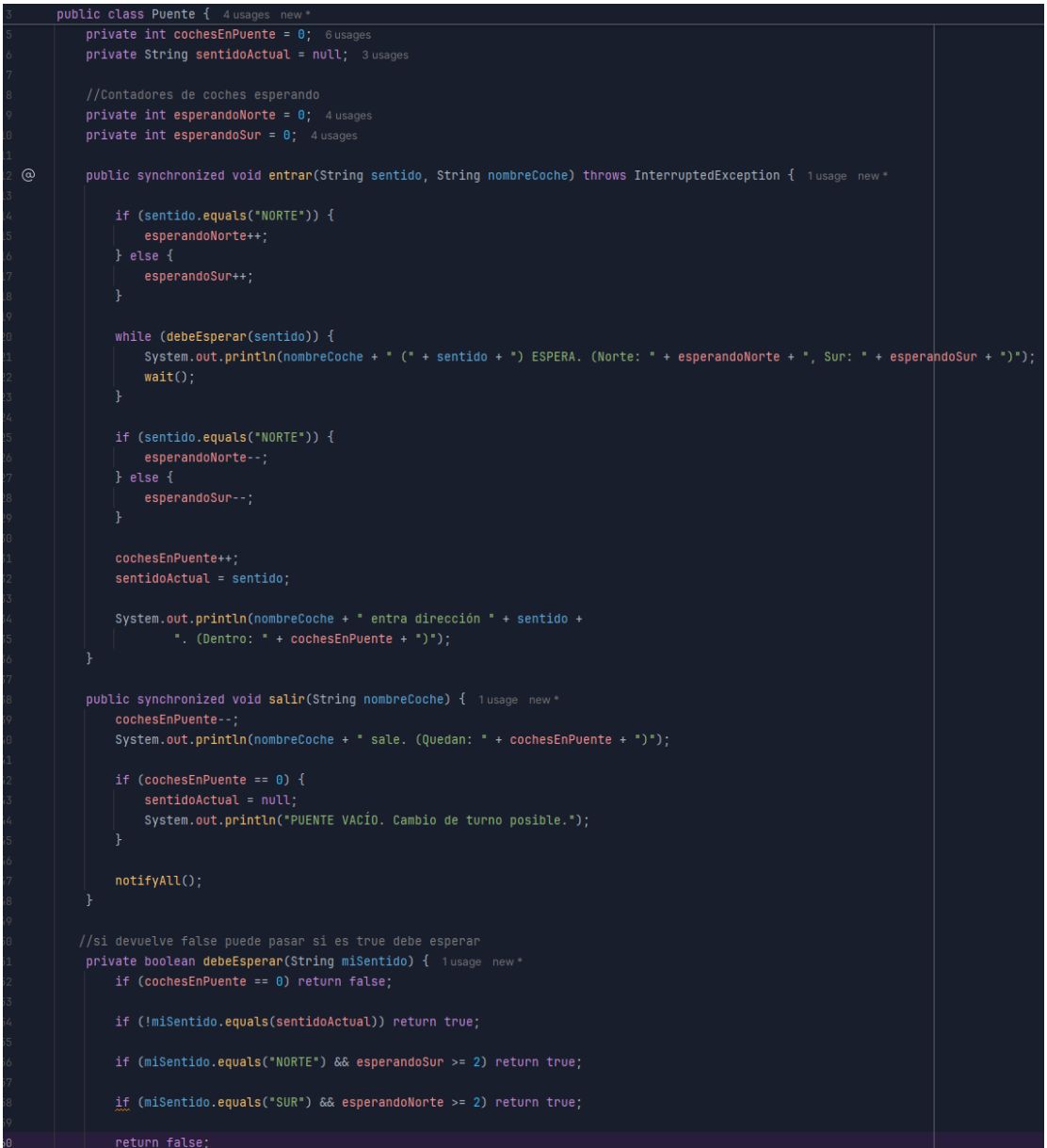
```
> Task :Ejercicio6.Main.main()
Coche-1 llega desde el NORTE.
Coche-1 entra al puente direcci◆n NORTE. (Coches dentro: 1)
Coche-2 llega desde el NORTE.
Coche-2 entra al puente direcci◆n NORTE. (Coches dentro: 2)
Coche-3 llega desde el NORTE.
Coche-3 entra al puente direcci◆n NORTE. (Coches dentro: 3)
Coche-4 llega desde el NORTE.
Coche-4 entra al puente direcci◆n NORTE. (Coches dentro: 4)
Coche-1 sale del puente. (Quedan: 3)
Coche-3 sale del puente. (Quedan: 2)
Coche-5 llega desde el SUR.
Coche-5 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-2 sale del puente. (Quedan: 1)
Coche-5 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-6 llega desde el SUR.
Coche-6 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-7 llega desde el NORTE.
Coche-7 entra al puente direcci◆n NORTE. (Coches dentro: 2)
Coche-8 llega desde el NORTE.
Coche-8 entra al puente direcci◆n NORTE. (Coches dentro: 3)
Coche-4 sale del puente. (Quedan: 2)
Coche-5 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-6 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-9 llega desde el NORTE.
Coche-9 entra al puente direcci◆n NORTE. (Coches dentro: 3)
Coche-10 llega desde el SUR.
Coche-10 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-8 sale del puente. (Quedan: 2)
Coche-5 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-6 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-10 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-7 sale del puente. (Quedan: 1)
Coche-5 (SUR) debe ESPERAR. Puente ocupado por NORTE
Coche-6 (SUR) debe ESPERAR. Puente ocupado por NORTE
```

Tomás Pérez Carrillo
2º DAM

Comentarios:

Para este ejercicio he implementado un Monitor (clase Puente) con métodos sincronizados. La lógica de concurrencia se basa en una condición de guarda dentro del bucle while: un vehículo debe detenerse (wait()) únicamente si el puente está ocupado (coches > 0) Y el sentido de circulación actual es contrario al suyo. Esto permite cumplir el requisito de eficiencia: si el puente está ocupado por coches del NORTE y llega otro del NORTE, este puede entrar directamente sin bloquearse, aprovechando el flujo. Solo se bloquea si el sentido es contrario. Al salir el último coche, se libera el sentido (sentidoActual = null) y se notifica a los hilos en espera.

7. En el ejercicio anterior, si estuvieran llegando coches continuamente por una misma dirección, los coches de la dirección contraria podrían quedar bloqueados indefinidamente. Modifica la solución para añadir la siguiente norma: Si en uno de los sentidos hay al menos dos coches esperando, no se permitirá que entren coches del sentido contrario al puente hasta que esos dos coches hayan cruzado.



```
public class Puente {  4 usages  new *
    private int cochesEnPuente = 0;  6 usages
    private String sentidoActual = null;  3 usages

    //Contadores de coches esperando
    private int esperandoNorte = 0;  4 usages
    private int esperandoSur = 0;  4 usages

    public synchronized void entrar(String sentido, String nombreCoche) throws InterruptedException {  1 usage  new *

        if (sentido.equals("NORTE")) {
            esperandoNorte++;
        } else {
            esperandoSur++;
        }

        while (debeEsperar(sentido)) {
            System.out.println(nombreCoche + " (" + sentido + ") ESPERA. (Norte: " + esperandoNorte + ", Sur: " + esperandoSur + ")");
            wait();
        }

        if (sentido.equals("NORTE")) {
            esperandoNorte--;
        } else {
            esperandoSur--;
        }

        cochesEnPuente++;
        sentidoActual = sentido;

        System.out.println(nombreCoche + " entra dirección " + sentido +
                ". (Dentro: " + cochesEnPuente + ")");
    }

    public synchronized void salir(String nombreCoche) {  1 usage  new *
        cochesEnPuente--;
        System.out.println(nombreCoche + " sale. (Quedan: " + cochesEnPuente + ")");

        if (cochesEnPuente == 0) {
            sentidoActual = null;
            System.out.println("PUENTE VACÍO. Cambio de turno posible.");
        }

        notifyAll();
    }

    //si devuelve false puede pasar si es true debe esperar
    private boolean debeEsperar(String miSentido) {  1 usage  new *
        if (cochesEnPuente == 0) return false;

        if (!miSentido.equals(sentidoActual)) return true;

        if (miSentido.equals("NORTE") && esperandoSur >= 2) return true;

        if (miSentido.equals("SUR") && esperandoNorte >= 2) return true;

        return false;
```

Tomás Pérez Carrillo
2º DAM

```java
public class Coche implements Runnable {  2 usages  new *

    private String nombre;  4 usages
    private String sentido; // "NORTE" o "SUR"  3 usages
    private Puente puente;  3 usages

    public Coche(String nombre, String sentido, Puente puente) {  1 usage  new *
        this.nombre = nombre;
        this.sentido = sentido;
        this.puente = puente;
    }

    @Override  new *
    public void run() {
        try {

            System.out.println(nombre + " llega desde el " + sentido + ".");

            puente.entrar(sentido, nombre);

            Thread.sleep( millis: (long) (Math.random() * 1000) + 500);

            puente.salir(nombre);

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```java
public class Main {  new *

    public static void main(String[] args) {  new *

        Puente puenteArchena = new Puente();

        String[] sentidos = {"NORTE", "SUR"};

        for (int i = 1; i <= 10; i++) {
            //Sentido al azar
            String sentidoAsignado = sentidos[(int)(Math.random() * 2)];

            Coche c = new Coche( nombre: "Coche-" + i, sentidoAsignado, puenteArchena);
            new Thread(c).start();

            // Pequeña pausa entre llegadas para que no lleguen todos en el milisegundo 0
            try {
                Thread.sleep((long) (Math.random() * 500));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Tomás Pérez Carrillo
2º DAM

Salida por terminal:

```
> Task :Ejercicio7.Main.main()
Coche-1 llega desde el NORTE.
Coche-1 entra direcci□n NORTE. (Dentro: 1)
Coche-2 llega desde el NORTE.
Coche-2 entra direcci□n NORTE. (Dentro: 2)
Coche-3 llega desde el SUR.
Coche-3 (SUR) ESPERA. (Norte: 0, Sur: 1)
Coche-4 llega desde el SUR.
Coche-4 (SUR) ESPERA. (Norte: 0, Sur: 2)
Coche-2 sale. (Quedan: 1)
Coche-3 (SUR) ESPERA. (Norte: 0, Sur: 2)
Coche-4 (SUR) ESPERA. (Norte: 0, Sur: 2)
Coche-5 llega desde el NORTE.
Coche-5 (NORTE) ESPERA. (Norte: 1, Sur: 2)
Coche-1 sale. (Quedan: 0)
PUENTE VAC□O. Cambio de turno posible.
Coche-3 entra direcci□n SUR. (Dentro: 1)
Coche-4 entra direcci□n SUR. (Dentro: 2)
Coche-5 (NORTE) ESPERA. (Norte: 1, Sur: 0)
Coche-3 sale. (Quedan: 1)
Coche-5 (NORTE) ESPERA. (Norte: 1, Sur: 0)
Coche-4 sale. (Quedan: 0)
PUENTE VAC□O. Cambio de turno posible.
Coche-5 entra direcci□n NORTE. (Dentro: 1)
Coche-5 sale. (Quedan: 0)
PUENTE VAC□O. Cambio de turno posible.

BUILD SUCCESSFUL in 3s
```

Tomás Pérez Carrillo
2º DAM

Comentarios:

El Ejercicio 7 soluciona el problema de inanición detectado en la versión anterior. He añadido dos contadores enteros (esperandoNorte y esperandoSur) que registran cuántos hilos están bloqueados en el wait(). La lógica crítica se encuentra en la condición de entrada: antes de acceder, el hilo incrementa su contador de espera. La condición del bucle while ahora evalúa dos escenarios para bloquearse:

1. Bloqueo por Seguridad: El puente tiene tráfico en sentido contrario.
2. Bloqueo por Prioridad (Justicia): Aunque el puente tenga tráfico en mi mismo sentido, si detecto que el contador de espera del sentido contrario supera el umbral establecido (2 coches), me bloqueo voluntariamente. Esto corta el flujo continuo de un solo sentido y fuerza la otra opción, permitiendo que el sentido contrario vacíe el puente y tome el control.

8. Una estación de carga rápida dispone de N puntos de carga, y un flujo constante de coches eléctricos llega para cargar sus baterías. Cada punto de carga solo puede atender a un coche a la vez. Debemos tener en cuenta que:

a. Los coches llegan de manera aleatoria y esperan a poder usar un punto de carga si todos están ocupados.

b. Cada coche permanece conectado durante un tiempo aleatorio para completar la carga.

c. Cuando termina, libera el punto de carga para que otro coche pueda usarlo.

d. Ningún coche puede cargar si no hay puntos de carga disponibles.

e. Todos los coches deben esperar su turno de manera justa.

f. La estación nunca sobrepasa la capacidad de puntos de carga.

Tomás Pérez Carrillo
2º DAM

```java
public class Estacion {  4 usages  new *

    private Semaphore puntosCarga;  4 usages
    private int capacidadTotal;  1 usage

    public Estacion(int numeroPuntos) {  1 usage  new *
        this.capacidadTotal = numeroPuntos;
        this.puntosCarga = new Semaphore(numeroPuntos, fair: true);
    }

    public void cargarBateria(String matricula) {  1 usage  new *
        try {

            System.out.println(matricula + " llega y espera turno...");

            puntosCarga.acquire(); // Resta 1 permiso. Si es 0, espera.

            int libres = puntosCarga.availablePermits();
            System.out.println(matricula + " ENTRA a cargar. (Sitios libres: " + libres + ")");

            //tiempo de carga simulado
            long tiempoCarga = (long) (Math.random() * 2000) + 1000;
            Thread.sleep(tiempoCarga);

            System.out.println(matricula + " termina de cargar (Tiempo: " + tiempoCarga + "ms). Sale de la estación.")

            puntosCarga.release();

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

}
```

```java
public class Coche implements Runnable {  2 usages  new *

    private String matricula;  2 usages
    private Estacion estacion;  2 usages

    public Coche(String matricula, Estacion estacion) {  1 us
        this.matricula = matricula;
        this.estacion = estacion;
    }

    @Override  new *
    public void run() {
        // El coche simplemente intenta cargar
        estacion.cargarBateria(matricula);
    }
}
```

Tomás Pérez Carrillo
2º DAM

```java
public class Main {  new *

    public static void main(String[] args) {  new *

        // estancia con 3 puntos de carga
        Estacion estacion = new Estacion( numeroPuntos: 3);

        System.out.println("ESTACIÓN DE CARGA (3 PUNTOS)");

        // Lanzamos 5 coches
        for (int i = 1; i <= 5; i++) {
            Coche c = new Coche( matricula: "Coche-" + i, estacion);
            new Thread(c).start();

            // Pequeña pausa para que lleguen escalonados y se aprecie el orden
            try {
                Thread.sleep( millis: 200);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Salida por terminal:

```
> Task :Ejercicio8.Main.main()
ESTACI🌐N DE CARGA (3 PUNTOS)
Coche-1 llega y espera turno...
Coche-1 ENTRA a cargar. (Sitios libres: 2)
Coche-2 llega y espera turno...
Coche-2 ENTRA a cargar. (Sitios libres: 1)
Coche-3 llega y espera turno...
Coche-3 ENTRA a cargar. (Sitios libres: 0)
Coche-4 llega y espera turno...
Coche-5 llega y espera turno...
Coche-1 termina de cargar (Tiempo: 1005ms). Sale de la estaci🌐n.
Coche-4 ENTRA a cargar. (Sitios libres: 0)
Coche-3 termina de cargar (Tiempo: 1630ms). Sale de la estaci🌐n.
Coche-5 ENTRA a cargar. (Sitios libres: 0)
Coche-2 termina de cargar (Tiempo: 2211ms). Sale de la estaci🌐n.
Coche-4 termina de cargar (Tiempo: 1454ms). Sale de la estaci🌐n.
Coche-5 termina de cargar (Tiempo: 1114ms). Sale de la estaci🌐n.

BUILD SUCCESSFUL in 3s
```

Tomás Pérez Carrillo
2º DAM

Comentarios:
Para este ejercicio he utilizado un Semaphore inicializado con el número de puntos de carga disponibles. Esta clase es ideal porque gestiona automáticamente los permisos, bloqueando a los coches cuando la estación está llena. Lo más importante ha sido activar el modo 'justo' (new Semaphore(N, true)), lo que asegura que los hilos se desbloqueen en el mismo orden en que llegaron (FIFO), cumpliendo el requisito de que los coches respeten su turno.

Comentarios finales sobre la tarea:

He de reconocer que la realización de esta tarea me ha supuesto un reto complejo. Por ello, he tenido que recurrir a diferentes fuentes para la realización de algunos ejercicios. Aún así, aunque haya necesitado esa ayuda adicional, he puesto mucho esfuerzo en analizar y entender el por qué de cada línea de código aquí escrita, con el objetivo principal de interiorizar los conceptos de cara al examen. Escribo esto ya que quiero ser totalmente transparente con el modus operandi que he seguido para la realización de la tarea.