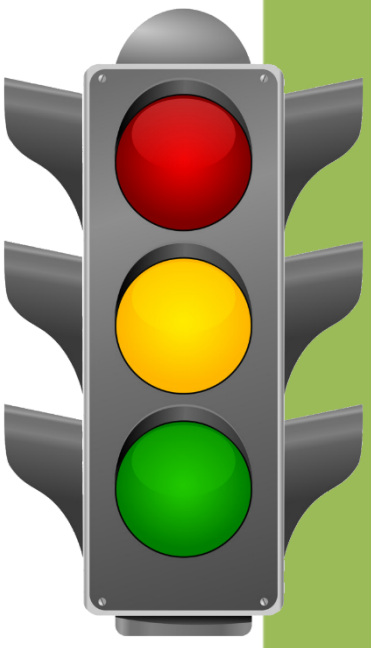
Desarrollo de Aplicaciones Multiplataforma

Informática y Comunicaciones

IES Valle del Jerte
Plasencia
informatica.iesvalledeljerteplasencia.es

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

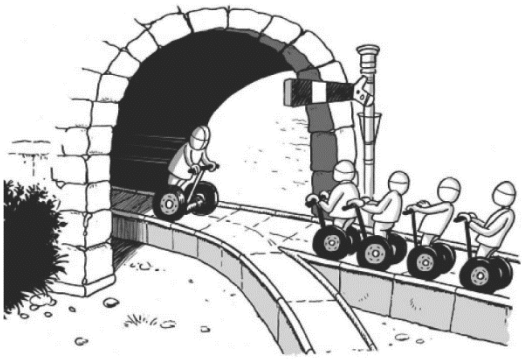
UT2: Semáforos en Java



2º DAM

Contenido

1. Introducción



El concepto de semáforos fue el primero en solucionar problemas de sincronización sin espera activa. Lo inventó **Edsger W. Dijkstra** a finales de la década de 1960. Está inspirado en las señales visuales ferroviarias que indican si un tren está habilitado para entrar en una vía. Es una construcción sencilla y eficiente que permite solucionar problemas genéricos de sincronización entre procesos.

Los semáforos se implementan habitualmente como servicios del núcleo de los sistemas operativos. Estos tienen la capacidad de bloquear y planificar la ejecución de los procesos e hilos.

Bloquear a un proceso hasta que pueda continuar su ejecución no necesita de funcionalidades adicionales sofisticadas, los sistemas operativos hacen lo mismo para todas las operaciones de entrada/salida.

En el caso de semáforos, los procesos se bloquean o ejecutan condicionados únicamente por el valor que tiene una variable entera. Una abstracción tan simple como potente.

1.1. Definición

Un semáforo es una estructura de datos cuyos valores asociados son enteros positivos, en el caso de los semáforos generales, o las constantes 0 y 1, si el semáforo es binario. De forma intuitiva, el valor 0 representa que el semáforo está cerrado (está en rojo) y cualquier valor positivo indica que el semáforo está abierto (está en verde). En este último caso, el valor concreto del semáforo representa el número de procesos que pueden pasar con el semáforo abierto. Estos valores del semáforo son modificados por una serie de operaciones atómicas.

En resumen:

Semáforo	Estructura de datos	contador	0-1 Semáforo binario
			0..N Semáforo general
	Operaciones	adquirir()	Intentamos adquirir el semáforo
		liberar()	Liberamos el semáforo

Tutorial mejor explicado

1.2. Semáforos en Java

Los semáforos en Java se representan como objetos de la clase `java.util.concurrent.Semaphore`.

Estructura:

```
public class UsingSemaphores {

    // creating a Semaphore
    Semaphore s = new Semaphore(10);
```

```
    // Multiple threads call this method on the same/shared instance of "UsingSemaphores"
    public void UsingSemaphore() throws InterruptedException {

        // acquire the permit
        s.acquire();

        // Code that runs inside
        // the acquired lock/permit

        // Release the permit you acquired
        s.release();

    }
}
```

Tutorial extenso en inglés (utiliza Java 8 con lambdas).
Documentación oficial

Un semáforo gestiona un contador interno que se decrementa en cada llamada a acquire() y se incrementa en cada llamada a release(). El contador nunca puede bajar de cero. Si una llamada a acquire() se encuentra el contador a cero, bloquea la ejecución del hilo en curso y queda a la espera de que otro hilo llame a release().

Otro tutorial sencillo

Constructor	Función
Semaphore (int permits)	Crea un `Semaphore` con el número dado de permisos (permits).
Semaphore (int permits, boolean fair)	Crea un `Semaphore` con el número dado de permisos y la configuración de justicia/imparcialidad .

Como podemos ver en el Constructor se puede indicar la justicia (fairness) del semáforo

- Si es justo (`fair` = true) entonces los hilos se desbloquean en orden de llegada (cola FIFO)
- Si no es justo (`fair` = false) entonces los hilos se desbloquean de forma aleatoria

Por defecto los semáforos en Java **no son justos** porque es más eficiente y en los usos normales no plantea problemas de inanición (starvation)
(Todos los métodos)

Método	Función
void acquire ()	Adquiere un permiso del semáforo, se bloquea si no hay permisos disponibles. Puede lanzar la excepción: InterruptedException
void acquire (int permits)	Adquiere varios permisos del semáforo, lo que se indiquen en la variable permits, se bloquea si no hay permisos disponibles. Puede lanzar la excepción: `InterruptedException`
int availablePermits()	Devuelve el número actual de permisos del semáforo.

`int getQueueLength()`	Devuelve una estimación del número de hilos que están esperando un permiso del semáforo
`boolean TryAcquire ()`	Adquiere un permiso del semáforo, solo si uno está disponible en el momento de la invocación. Es decir, si hay permisos devuelve true y decrementa los permisos del semáforo y si no hay permisos, devuelve false inmediatamente
`void release ()`	Libera un permiso y lo devuelve al semáforo.
`void release (int permits)`	Libera el número dado de permisos y los devuelve al semáforo.
`String toString()`	Devuelve una cadena que identifica el semáforo, así como su estado.
`void reducePermits(int permits)`	Reduce el número de permisos disponibles por la reducción indicada.

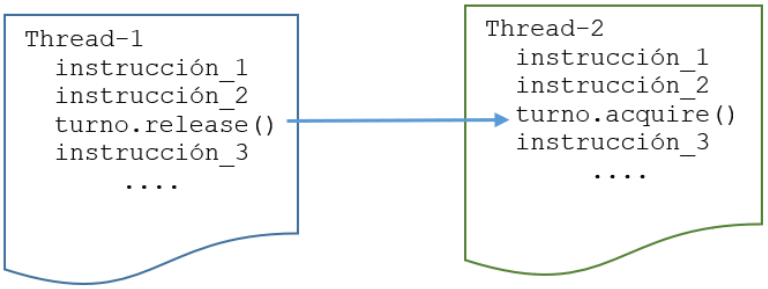
Ten en cuenta que si tengo un semáforo con permisos 0 y hago con release(10), el semáforo pasaría a tener 10 permisos:

```
Semaphore semaphore = new Semaphore(0);
semaphore.release(10);
System.out.println("Permisos:"+ semaphore.availablePermits());// Permisos: 10
```

Semáforos: Sincronización entre hilos

Si nos declaramos el semáforo con permisos 0, el primer hilo que haga la llamada a `acquire()`, quedará bloqueado hasta que haya permisos, es decir, hasta que llamemos al método `release()`:

```
Semaphore turno= new Semaphore(0);
```



Ejemplo 1:

Vamos a crearnos un programa que pinte por pantalla "BUENAS TARDES", para ello nos crearemos 2 hilos:

- HiloBuenas hace una espera de 1000 milisegundos y muestra por pantalla: "BUENAS".
- HiloTardes, no hace ninguna espera y muestra por pantalla: "TARDES".

Se trata de mostrar en orden las palabras "**BUENAS TARDES**" utilizando Semáforos.

```
package ejemplo1;
import java.util.concurrent.Semaphore;

public class HiloTardes extends Thread {
	Semaphore semaforo;// referencia a un semáforo común a otros hilos

	public HiloTardes(Semaphore semaforo) {
```

```
			this.semaforo = semaforo;
		}

		@Override
		public void run() {

			System.out.println("HiloTardes:numero de permisos del semáforo: " +
													semaforo.availablePermits());
			try {
			semaforo.acquire();//cogemos un permiso del semáforo, si no hay
								se bloquea
			} catch (InterruptedException e) {
				e.printStackTrace();
			}

			System.out.println("TARDES");

		}

}
```

```
package ejemplo1;

import java.util.concurrent.Semaphore;

public class HiloBuenas extends Thread {

		Semaphore semaforo;//referencia a un semaforo común a otros hilos

		public HiloBuenas(Semaphore semaforo) {
			this.semaforo = semaforo;
		}

		@Override
		public void run() {
			try {
				sleep(1000);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}

			System.out.println("BUENAS");
			semaforo.release();//liberamos un permiso del semáforo

			System.out.println("HiloBuenas:numero de permisos del semáforo: "+
													semaforo.availablePermits());

		}

}
```

```
package ejemplo1;
```

```
import java.util.concurrent.Semaphore;


public class Principal {
	public static void main(String[] args) {

		Semaphore semaforo = new Semaphore(0);
		//inicializamos el semáforo con 0 permisos, para bloquear el
		// hilo que llame a acquire

		HiloBuenas hiloBuenas = new HiloBuenas(semaforo);
		HiloTardes hiloTardes = new HiloTardes(semaforo);

		hiloTardes.start();
		hiloBuenas.start();



	}
}
```

ACTIVIDADES

1. Crea dos clases que implementen la Interfaz Runnable. La primera clase se llamará hiloTIC y la segunda hiloTAC:
   - La clase hiloTIC estará continuamente pintando por pantalla TIC.
   - La clase hiloTAC estará continuamente pintando por pantalla TAC.

   Coordina ambas clases utilizando semáforos para que la salida final del programa sea intercalada:

   -- TIC-TAC-TIC-TAC… El hilo TIC es el que empieza el ciclo

2. Crea un programa en java que conste de 30 HiloIniciales y 1 HiloFinal:

   - Los 30 HiloInicial pintan por pantalla su identificador (los identificadores van del 1 al 30). Estos hilos no están sincronizados por lo que el 27 puede que pinte su identificador antes que el 30.
   - El hiloFinal pinta en su ejecución: “Hola soy el hilo final”

   HiloFinal no podrá escribir su nombre hasta que todos los otros hilos anteriores no hayan pintado su identificador por pantalla.

Extra: Crea un escenario simple donde múltiples hilos representan coches intentando entrar a un párking. El párking tiene un número de aparcamientos limitado, y solo un cierto número de coches entran a la vez. utiliza semáforos para controlar el acceso al párking.

```
import java.util.concurrent.Semaphore;

class ParkingLot {
```

```
    private Semaphore parkingSpaces;

    public ParkingLot(int spaces) {
        this.parkingSpaces = new Semaphore(spaces);
    }

    public void enterParkingLot(String carName) {
        try {
            System.out.println(carName + " is trying to enter the parking lot.");
            parkingSpaces.acquire(); // El coche intenta adquirir espacio.

            // Car has successfully acquired a parking space
            System.out.println(carName + " has entered the parking lot.");
            Thread.sleep(2000); // Simulación del coche aparcado cierto tiempo.

            // El coche está saliendo del aparcamiento
            System.out.println(carName + " is leaving the parking lot.");
            parkingSpaces.release(); // El coche se va, dejando libre el aparcamiento de nuevo.
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class Car extends Thread {
    private String carName;
    private ParkingLot parkingLot;

    public Car(String carName, ParkingLot parkingLot) {
        this.carName = carName;
        this.parkingLot = parkingLot;
    }

    public void run() {
        parkingLot.enterParkingLot(carName);
    }
}

public class SemaphoreParkingLotExample {
    public static void main(String[] args) {
        int numberOfSpaces = 3; // Número de aparcamientos
        ParkingLot parkingLot = new ParkingLot(numberOfSpaces);

        // Simulación de múltiples coches tratando de entrar al parking a la vez
        for (int i = 1; i <= 5; i++) {
            String carName = "Car " + i;
            Car car = new Car(carName, parkingLot);
            car.start(); // Lanzamos cada hilo
        }
    }
}
```

Extra2: Simula un escenario donde múltiples hilos representan estudiantes intentando usar impresoras en un aula. Hay un número limitado de impresoras, y sólo un cierto número de estudiantes pueden utilizar impresoras a la vez. Usa semáforos para controlar el acceso a las impresoras.

1.2.1. Semáforos: Limitar el número de hilos en la región crítica

El número de hilos que pueden entrar en la región crítica se especifican a la hora de instanciar el semáforo.

```
Semaphore mutex= new Semaphore(5);//máximo 5 hilos
……
```

```
mutex.acquire();
//región critica
mutex.release();
……..
```

CONSEJOS:

- A los semáforos binarios para el acceso exclusivo a variables compartidas se les suele denominar **mutex** (*mutual exclusive*).
- Utiliza un semáforo sin permisos para bloquear a aquellos hilos que tengan que esperar (**`wait()`**) mientras no se cumpla una determinada condición. Utiliza un semáforo por cada condición de bloqueo.
- Abre el semáforo *mutex* para evitar interbloqueos siempre que sea posible, sobre todo antes de abrir/cerrar otro semáforo.
- En los semáforos no existe un **`notifyAll()`** como en los monitores, tendrás que llamar tantas veces a **`release()`** como veces hayas llamado a **`acquire()`.**

Ejemplo 2: 

Nos han pedido desarrollar una aplicación para gestionar el nuevo Parking que se va a construir en Plasencia en la calle Velázquez. El parking tiene una capacidad máxima de 2 plazas y tiene una única entrada/salida dónde hay una barrera que gestiona la entrada y salida de los coches al parking, y un cartel luminoso dónde se informa de las plazas libres disponibles. Si en el parking hay plazas libres el coche podrá entrar a buscar aparcamiento, pero si el parking está lleno el coche deberá esperar a que se libere alguna plaza.

```
package ejemplo2_v1;

public class Coche implements Runnable {

	Parking parking;
	int numcoche;

	public int getNumcoche() {
		return numcoche;
	}

	public void setNumcoche(int numcoche) {
		this.numcoche = numcoche;
	}

	public Coche(Parking parking, int numcoche) {
		this.parking = parking;
		this.numcoche = numcoche;

	}

	@Override
	public void run() {
		System.out.println("Arranca el coche número " + getNumcoche());

		try {
			// el coche sale de su casa y se da una vuelta por Plasencia
			Thread.sleep((long) Math.random() * 40000);
```

```
                    System.out.println("\t||El coche número " + getNumcoche() + "
                                                llega al parking e intenta entrar");
                    parking.entrarParking(this);
                    // encuentra aparcamiento y se va de compras
                    Thread.sleep((long) Math.random() * 90000);

                    parking.salirParking(this);



             } catch (InterruptedException e) {

                    e.printStackTrace();
             }

      }



}
```

```
package ejemplo2_v1;

import java.util.concurrent.Semaphore;

public class Parking {

      static final int CAPACIDAD = 2;
      // nos declaramos el semáforo con 2 permisos y con justicia (FIFO)
      private Semaphore semaforo = new Semaphore(CAPACIDAD, true);

      public void entrarParking(Coche coche) {

             try {
                    semaforo.acquire();
                    System.out.println("\t\t=>El coche número " +
                     coche.getNumcoche() + " entra al Parking. Plazas libres:"
                                                 + semaforo.availablePermits());
                    System.out.println("Coches esperando para entrar: " +
                                                      semaforo.getQueueLength());

             } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
             }
      }

      public void salirParking(Coche coche) {

             System.out.println("\t<=El coche número " + coche.getNumcoche() + "
                                                sale del Parking.Plazas libres:"
                                                 + semaforo.availablePermits());

             semaforo.release();

      }
}
```

```
package ejemplo2_v1;

public class Principal {

	public static void main(String[] args) {

	Parking parking= new Parking();
	for (int i = 1; i <= 5; i++) {

		Coche coche= new Coche(parking,i);
		Thread th= new Thread(coche);
		th.start();

	}

	}

}
```

Una de las posibles salidas por pantalla sería:

```
Arranca el coche número 1
Arranca el coche número 2
Arranca el coche número 3
Arranca el coche número 4
Arranca el coche número 5
	||El coche número 4 llega al parking e intenta entrar
	||El coche número 2 llega al parking e intenta entrar
	||El coche número 5 llega al parking e intenta entrar
	||El coche número 3 llega al parking e intenta entrar
	||El coche número 1 llega al parking e intenta entrar
		=>El coche número 5 entra al Parking. Plazas libres:1
Coches esperando para entrar: 2
		=>El coche número 2 entra al Parking. Plazas libres:0
Coches esperando para entrar: 2
	<=El coche número 5 sale del Parking.Plazas libres:0
	<=El coche número 2 sale del Parking.Plazas libres:0
		=>El coche número 3 entra al Parking. Plazas libres:1
Coches esperando para entrar: 2
		=>El coche número 1 entra al Parking. Plazas libres:0
Coches esperando para entrar: 1
	<=El coche número 3 sale del Parking.Plazas libres:0
	<=El coche número 1 sale del Parking.Plazas libres:0
		=>El coche número 4 entra al Parking. Plazas libres:0
Coches esperando para entrar: 0
	<=El coche número 4 sale del Parking.Plazas libres:1
```

ACTIVIDADES

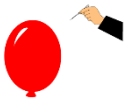
3. A partir del ejemplo 2 (Parking), realiza las siguientes modificaciones:
   - Las plazas del Parking están numeradas, y nos interesa saber qué plaza ocupa cada coche, en cada momento. La plaza que quede libre será ocupada por el siguiente que entre. En un principio todas las plazas están libres.
   - Cada vez que entre y salga un coche debemos mostrar actualizadas las plazas que quedan disponibles.
   - Utiliza las técnicas que creas más adecuadas para mostrar de forma lógica y ordenada los mensajes de entrada-salida de los coches y el número de plazas disponibles.

A continuación, se muestra una posible salida del ejercicio, donde el Parking tiene únicamente 2 plazas y hay 4 coches deseando aparcar:

```
Arranca el coche número 2
Arranca el coche número 3
Arranca el coche número 1
Arranca el coche número 4
||El coche número 1 llega al parking e intenta entrar
||El coche número 3 llega al parking e intenta entrar
||El coche número 2 llega al parking e intenta entrar
=>El coche número 3 entra al Parking
=>El coche número 1 entra al Parking
||El coche número 4 llega al parking e intenta entrar
---------Coche número 3 ha aparcado en la plaza 0.Quedan libres 1
---------Coche número 1 ha aparcado en la plaza 1.Quedan libres 0
<= Coche número 3 abandona el parking.Quedan libres 1
<= Coche número 1 abandona el parking.Quedan libres 2
=>El coche número 2 entra al Parking
---------Coche número 2 ha aparcado en la plaza 0.Quedan libres 1
=>El coche número 4 entra al Parking
<= Coche número 2 abandona el parking.Quedan libres 2
---------Coche número 4 ha aparcado en la plaza 0.Quedan libres 1
<= Coche número 4 abandona el parking.Quedan libres 2
```

Otra modificación: añade un mensaje para los coches que se quedan esperando.

Imagina 3 amigos que quieren explotar 1000 globos. Cada uno de estos amigos va a intentar explotar tantos globos como sea capaz.



Realiza un programa en el que se comparta una variable “explotados” por los amigos y mediante semáforos (está prohibido utilizar synchronized en todo el código del programa), controlar que el total de globos explotados sea 1000.

Al final de la ejecución muestra por pantalla qué cantidad de globos ha explotado cada uno.

4. Que el instituto está pasando por un momento económico crítico es algo que nadie duda. Esta vez los recortes afectan a los profesores. En concreto al espacio en común que ellos comparten, la sala de profesores. Esta sala se va a reducir, lo que va a imposibilitar que haya más de 3 profesores dentro de la sala a la vez.

Sincroniza mediante semáforos la ejecución para asegurarte de que nunca va a haber más de 3 profesores en la sala teniendo en cuenta que el número de profesores totales es 7. Pinta cuando los profesores entran en la sala y cuando se van a disponer a salir. Todos los mensajes deben estar en la zona sincronizada.

NOTA: Los profesores hacen una espera de 200 ms dentro de la sala de profesores. También hacen una espera de 20 ms cuando salen antes de volver a entrar.

Posible salida por pantalla:

```
Encarni se dirige a la Sala de Profesores
Bacterio se dirige a la Sala de Profesores
La de mates se dirige a la Sala de Profesores
        +++ Bacterio está dentro de la Sala de Profesores. En estos momentos hay [1]
        +++ La de mates está dentro de la Sala de Profesores. En estos momentos hay [2]
Eusebio se dirige a la Sala de Profesores
Topo Gigio se dirige a la Sala de Profesores
        +++ Encarni está dentro de la Sala de Profesores. En estos momentos hay [3]
Rompetechos se dirige a la Sala de Profesores
El gallego se dirige a la Sala de Profesores
                ---Bacterio abandona la Sala de Profesores.En estos momentos hay [2]
        +++ Eusebio está dentro de la Sala de Profesores. En estos momentos hay [3]
                ---La de mates abandona la Sala de Profesores.En estos momentos hay [2]
                ---Encarni abandona la Sala de Profesores.En estos momentos hay [1]
        +++ Topo Gigio está dentro de la Sala de Profesores. En estos momentos hay [2]
        +++ El gallego está dentro de la Sala de Profesores. En estos momentos hay [3]
Bacterio se dirige a la Sala de Profesores
La de mates se dirige a la Sala de Profesores
Encarni se dirige a la Sala de Profesores
                ---Eusebio abandona la Sala de Profesores.En estos momentos hay [2]
        +++ Rompetechos está dentro de la Sala de Profesores. En estos momentos hay [3]
                ---Topo Gigio abandona la Sala de Profesores.En estos momentos hay [2]
                ---El gallego abandona la Sala de Profesores.En estos momentos hay [1]
        +++ Bacterio está dentro de la Sala de Profesores. En estos momentos hay [2]
        +++ La de mates está dentro de la Sala de Profesores. En estos momentos hay [3]
```

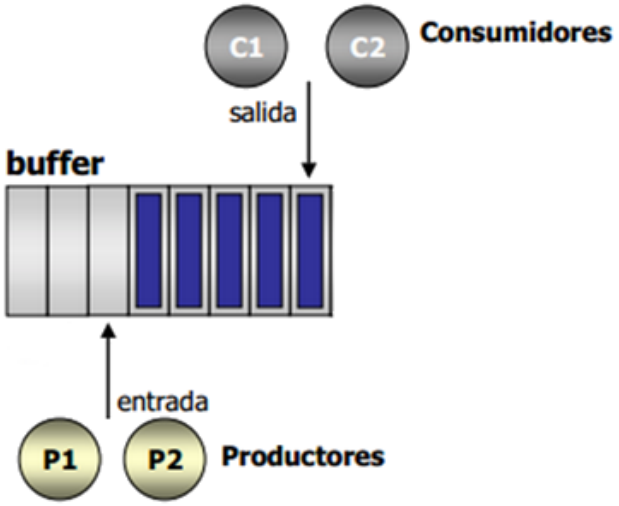
5. En una tienda de pájaros están teniendo problemas para tener a todos sus canarios felices. Tenemos 10 canarios que comparten una jaula en la que hay un único plato con alpiste y un columpio para hacer ejercicio. Todos los canarios quieren inicialmente comer del plato (tardan 5sg) y, después columpiarse (tardan 6sg). Pero se encuentran con el inconveniente de que sólo 3 de ellos pueden comer del plato al mismo tiempo y sólo 1 puede columpiarse. Desarrolla una aplicación en la que se ejecuten los canarios concurrentemente de forma que sincronicen sus actividades usando <u>únicamente semáforos</u>, no utilices más variables. *(1 punto)*

Posible salida por pantalla:

```
Canario-2 está comiendo alpiste <O)
Canario-3 está comiendo alpiste <O)
Canario-1 está comiendo alpiste <O)
        Canario-2 terminó de comer
        Canario-1 terminó de comer
                Canario-2 está en el columpio ||__||
        Canario-3 terminó de comer
Canario-5 está comiendo alpiste <O)
Canario-0 está comiendo alpiste <O)
Canario-4 está comiendo alpiste <O)
        Canario-5 terminó de comer
        Canario-0 terminó de comer
        Canario-4 terminó de comer
Canario-7 está comiendo alpiste <O)
Canario-6 está comiendo alpiste <O)
Canario-8 está comiendo alpiste <O)
                Canario-2 terminó de columpiarse
                Canario-1 está en el columpio ||__||
        Canario-7 terminó de comer
        Canario-8 terminó de comer
        Canario-6 terminó de comer
Canario-9 está comiendo alpiste <O)
                Canario-1 terminó de columpiarse
                Canario-3 está en el columpio ||__||
        Canario-9 terminó de comer
                Canario-3 terminó de columpiarse
                Canario-5 está en el columpio ||__||
                Canario-5 terminó de columpiarse
                Canario-0 está en el columpio ||__||
                Canario-0 terminó de columpiarse
                Canario-4 está en el columpio ||__||
                Canario-4 terminó de columpiarse
                Canario-7 está en el columpio ||__||
                Canario-7 terminó de columpiarse
                Canario-8 está en el columpio ||__||
                Canario-8 terminó de columpiarse
                Canario-6 está en el columpio ||__||
                Canario-6 terminó de columpiarse
                Canario-9 está en el columpio ||__||
                Canario-9 terminó de columpiarse
```

Ejemplo 3: Productor-Consumidor

Volvemos al ejemplo del Productor-Consumidor, dónde tenemos un buffer que vamos a implementar mediante una Cola (Queue) en Java. Las Colas son un tipo de Lista dónde sus elementos se introducen (encolan) únicamente por el final de la Cola y se extraen únicamente por el extremo contrario, denominado frente o principio de la Cola. Las colas también se conocen como estructuras *FIFO (First Input First Ouput)*



Los métodos de **`Queue`** que vamos a utilizar son:

Métodos	Descripción
`boolean add(E e)`	Inserta el elemento especificado (E) en la Cola. Si ha podido añadir devuelve true y si no devuelve la excepción `RuntimeException.IllegalStateException`.
`boolean offer(E e)`	Inserta el elemento especificado (E) en la Cola. Si ha podido añadir devuelve true y si no false. (**Preferible)**
`E poll()`	Recupera y elimina el elemento que está en el frente de la Cola devuelve null si la Cola está vacía. (**Preferible)**
`E remove()`	Recupera y elimina el elemento que está en el frente de la Cola, si la Cola está vacía devuelve la excepción `RuntimeException.NoSuchElementException`

Vamos a ver el código del Productor-Consumidor:

```
import java.util.LinkedList;
import java.util.Queue;

public class Buffer {
	static Queue<String> cola = new LinkedList<String>();


	public static void mostrarCola() {
		System.out.println("------------------------------");
		//pasamos la cola a un array para poder mostrarlo al revés
		String[] arr = cola.toArray(new String[cola.size()]);
		//recorremos el array al revés
		for (int i = arr.length - 1; i >= 0; i--) {
			System.out.print("| " + arr[i] + " ");
		}

		System.out.println("\n------------------------------\n");
	}

}
```

```
import java.util.concurrent.Semaphore;

public class Productor extends Thread {
	String name;
	Semaphore vacio; //cola para esperar cuando está vacío
	Semaphore lleno; //cola para esperar cuando está lleno
	Semaphore mutex;


	public Productor(String name, Semaphore vacio, Semaphore lleno, Semaphore mutex) {
		this.name = name;
		this.vacio = vacio;
		this.lleno = lleno;
		this.mutex = mutex;
	}

	@Override
	public void run() {
		//vamos a repetir el proceso 5 veces (producimos 5 números cada vez)
		for (int i = 0; i < 5; i++) {

		 try {
		     lleno.acquire();//tomamos un permiso del semáforo (inicialmente son 5)
		     mutex.acquire();//para acceder en exclusión mutua a la cola
		     } catch (InterruptedException e) {
		              e.printStackTrace();
		       }

		     String produce=name+"-"+i;
		     //insertamos un elemento en la cola
		     Buffer.cola.offer(produce);
		     System.out.println("Productor "+name+" produce "+ produce);
		     Buffer.mostrarCola();
		     mutex.release();//salimos de la exclusión mutua
		     vacio.release();//liberamos un permiso de vacío, avisamos que hay un
		                                             elemento,inicialmente está a 0


		}
	}

}
```

```
import java.util.concurrent.Semaphore;

public class Consumidor extends Thread {
	String name;
	Semaphore vacio;
	Semaphore lleno;
	Semaphore mutex;

	public Consumidor(String name, Semaphore vacio, Semaphore lleno, Semaphore mutex) {
		this.name = name;
		this.vacio = vacio;
		this.lleno = lleno;
		this.mutex = mutex;
	}

	@Override
	public void run() {
		//cada consumidor consume 5 productos
	for (int i = 0; i < 5; i++) {
		try {
		vacio.acquire();//tomamos un permiso del semáforo, como no tiene se bloquea
                                                        hasta que el productor produzca
		mutex.acquire();//para acceder en exclusión mutua a la cola
		  } catch (InterruptedException e) {
				e.printStackTrace();
		}

		String consumido=Buffer.cola.poll();//tomamos un valor de la cola y se borra
		System.out.println("Consumidor "+name+"==>"+consumido);
		Buffer.mostrarCola();
		mutex.release();//salimos de la exclusión mutua
		lleno.release();//liberamos un permiso del semáforo lleno
		yield();
		}
	}
}
```

```
import java.util.concurrent.Semaphore;

public class Main {

public static void main(String[] args) {
  Semaphore vacio = new Semaphore(0);//semáforo sin permisos, el primero que llame
                                                        a acquire() se bloqueará
  Semaphore lleno = new Semaphore(5); //semáforo que se bloqueará cuando el buffer
                                                                esté lleno
  Semaphore mutex = new Semaphore(1); //semáforo para la sección crítica.

  Consumidor [] consumidores = new Consumidor[2];
  Productor [] productores = new Productor[2];

for (int i = 0; i < consumidores.length; i++) {
	consumidores[i] = new Consumidor("C"+(i+1), vacio, lleno, mutex);
	consumidores[i].start();
	}

for (int i = 0; i < productores.length; i++) {
	productores[i] = new Productor("P"+(i+1), vacio, lleno, mutex);
	productores[i].start();
	}
  }
}
```

Una de las posibles ejecuciones sería:

```
Productor P1 produce P1-0
-----------------------------
| P1-0
-----------------------------

Productor P2 produce P2-0
-----------------------------
| P2-0 | P1-0
-----------------------------

Consumidor C1==>P1-0
-----------------------------
| P2-0
-----------------------------

Consumidor C2==>P2-0
-----------------------------

-----------------------------

Productor P2 produce P2-1
-----------------------------
| P2-1
-----------------------------

Productor P1 produce P1-1
-----------------------------
| P1-1 | P2-1
-----------------------------

Consumidor C2==>P2-1
-----------------------------
| P1-1
-----------------------------

Consumidor C1==>P1-1
-----------------------------

-----------------------------
```

```
Productor P2 produce P2-2
-----------------------------
| P2-2
-----------------------------

Productor P1 produce P1-2
-----------------------------
| P1-2 | P2-2
-----------------------------

Consumidor C2==>P2-2
-----------------------------
| P1-2
-----------------------------

Productor P2 produce P2-3
-----------------------------
| P2-3 | P1-2
-----------------------------

Consumidor C1==>P1-2
-----------------------------
| P2-3
-----------------------------

Productor P1 produce P1-3
-----------------------------
| P1-3 | P2-3
-----------------------------

Consumidor C2==>P2-3
-----------------------------
| P1-3
-----------------------------

Productor P2 produce P2-4
-----------------------------
| P2-4 | P1-3
-----------------------------
```

```
Consumidor C1==>P1-3
-----------------------------
| P2-4
-----------------------------

Consumidor C2==>P2-4
-----------------------------

-----------------------------

Productor P1 produce P1-4
-----------------------------
| P1-4
-----------------------------

Consumidor C1==>P1-4
-----------------------------

-----------------------------
```

ACTIVIDADES

6. A partir del ejemplo 3 (Productor-Consumidor), realiza los cambios necesarios, para que los procesos Productor escriban una cadena en un fichero de texto y desde ese mismo fichero los procesos Consumidor lean la cadena escrita.

7. Tenemos un baño público en el que cabe un número ilimitado de personas. Cada cierto tiempo un empleado realiza la limpieza del baño, actuando de la siguiente forma: pone un letrero para impedir que entren personas al baño, se espera a que quienes están dentro acaben y salgan, y una vez que el recinto está vacío, realiza la limpieza. Cuando termina de limpiar, retira el letrero y se marcha, quedando el baño libre para que entren nuevos usuarios. Se quiere implementar una aplicación que simule este baño, con hilos que implementan a las personas que interactúan con él.

A continuación, se muestran los bocetos de los procedimientos que ejecutarán los hilos personas y el hilo limpiador:

```
void usar_baño () {
  … esperar a que el baño esté libre
  … y llegados a este punto…
  … hacer lo habitual en un baño
  … y salir del baño
}
```

```
void limpiar_baño() {
  … poner letrero «baño cerrado»
  … esperar a que se quede vacío
  … limpiar el baño
  … quitar letrero
}
```

Posible ejecución del programa con 10 personas entrando solamente una vez en el baño y 1 limpiador, limpiando únicamente una vez el baño.

```
Ana quiere entrar en el baño
Natalia quiere entrar en el baño
Victor quiere entrar en el baño
Jesus quiere entrar en el baño
Sergio quiere entrar en el baño
Pepe quiere entrar en el baño
        Ana está utilizando el baño
Luis quiere entrar en el baño
        Luis está utilizando el baño
Claudia quiere entrar en el baño
        Natalia está utilizando el baño
        Victor está utilizando el baño
--El limpiador pone el cartel para limpiar el baño
        XX Jesus está esperando a que terminen de limpiar
        XX Claudia está esperando a que terminen de limpiar
        XX Pepe está esperando a que terminen de limpiar
        XX Sergio está esperando a que terminen de limpiar
                =>Ana abandona el baño
                =>Luis abandona el baño
                =>Victor abandona el baño
                =>Natalia es el/la último/a en abandonar el baño, avisa al limpiador
        XX Limpiando baño
                 Baño limpio
        Jesus está utilizando el baño
        Sergio está utilizando el baño
        Claudia está utilizando el baño
        Pepe está utilizando el baño
                =>Jesus abandona el baño
                =>Pepe abandona el baño
                =>Claudia abandona el baño
                =>Sergio abandona el baño
```

8. Queremos desarrollar una aplicación que controle el acceso a una sala de baile en el que hay que procurar que entre un número equitativo de hombres y de mujeres. Cuando una persona intenta entrar en el salón y hay más personas de su sexo que del contrario, esta persona queda esperando hasta que salgan personas de su mismo sexo y se equilibre el balance.

   Implementa mediante semáforos la gestión de la sala de baila, introduce las variables auxiliares que consideres necesarias.

```
Hombre-0 quiere entrar en el salón de baile [ M:0, H:0]
        Hombre-0 está bailando [ M:0, H:1]
Mujer-0 quiere entrar en el salón de baile [ M:0, H:1]
        Mujer-0 está bailando [ M:1, H:1]
Mujer-1 quiere entrar en el salón de baile [ M:1, H:1]
        Mujer-1 está bailando [ M:2, H:1]
Hombre-1 quiere entrar en el salón de baile [ M:2, H:1]
        Hombre-1 está bailando [ M:2, H:2]
Mujer-2 quiere entrar en el salón de baile [ M:2, H:2]
        Mujer-2 está bailando [ M:3, H:2]
Hombre-2 quiere entrar en el salón de baile [ M:3, H:2]
        Hombre-2 está bailando [ M:3, H:3]
Hombre-3 quiere entrar en el salón de baile [ M:3, H:3]
        Hombre-3 está bailando [ M:3, H:4]
Hombre-5 quiere entrar en el salón de baile [ M:3, H:4]
        Hombre-5 esperando como un burro en la puerta del baile
Hombre-4 quiere entrar en el salón de baile [ M:3, H:4]
        Hombre-4 esperando como un burro en la puerta del baile
Mujer-3 quiere entrar en el salón de baile [ M:3, H:4]
        Mujer-3 está bailando [ M:4, H:4]
                Mujer-0 terminó de bailar[ M:3, H:4]
                Mujer-1 terminó de bailar[ M:2, H:4]
                Hombre-1 terminó de bailar[ M:2, H:3]
                Mujer-2 terminó de bailar[ M:1, H:3]
                Hombre-2 terminó de bailar[ M:1, H:2]
                Hombre-3 terminó de bailar[ M:1, H:1]
        Hombre-5 está bailando [ M:1, H:2]
                Mujer-3 terminó de bailar[ M:0, H:2]
                Hombre-0 terminó de bailar[ M:0, H:1]
                Hombre-5 terminó de bailar[ M:0, H:0]
        Hombre-4 está bailando [ M:0, H:1]
                Hombre-4 terminó de bailar[ M:0, H:0]
```

9. En una tienda de pájaros tenemos una jaula con 10 canarios que revolotean en su interior. Regularmente (cada 10ms) cada canario quiere comer de un comedero de alpiste, en el que solamente puede haber 3 canarios comiendo al mismo tiempo. Si un canario quiere comer y el comedero está ocupado, se debe esperar a que haya hueco. Por su parte, el encargado de la tienda de vez en cuando (cada 10ms) repone el alpiste del comedero. Mientras el encargado está reponiendo, ningún canario puede estar comiendo: el encargado debe

esperar a que no haya canarios en el comedero y, una vez libre, ningún canario entra a comer hasta que el encargado termina de reponer.

Resuelve este ejercicio utilizando semáforos para que los canarios y el encargado se sincronicen entre todos ellos conforme al enunciado expuesto.
Posible salida por pantalla:

```
-----Canario 1 está comiendo alpiste <O)
-----Canario 2 está comiendo alpiste <O)
-----Canario 0 está comiendo alpiste <O)
El encargado quiere reponer alpiste
        Canario 2 abandona el comedero WW_O_WW
        Canario 1 abandona el comedero WW_O_WW
        Canario 0 abandona el comedero WW_O_WW
+++++Reponiendo alpiste
-----Canario 8 está comiendo alpiste <O)
-----Canario 7 está comiendo alpiste <O)
-----Canario 9 está comiendo alpiste <O)
        Canario 8 abandona el comedero WW_O_WW
-----Canario 3 está comiendo alpiste <O)
        Canario 9 abandona el comedero WW_O_WW
        Canario 7 abandona el comedero WW_O_WW
-----Canario 5 está comiendo alpiste <O)
-----Canario 6 está comiendo alpiste <O)
El encargado quiere reponer alpiste
        Canario 3 abandona el comedero WW_O_WW
        Canario 5 abandona el comedero WW_O_WW
        Canario 6 abandona el comedero WW_O_WW
+++++Reponiendo alpiste
-----Canario 0 está comiendo alpiste <O)
-----Canario 1 está comiendo alpiste <O)
-----Canario 2 está comiendo alpiste <O)
        Canario 2 abandona el comedero WW_O_WW
```

10. Tenemos un depósito de agua con una capacidad de 200 litros. Sobre este depósito se han definido dos operaciones, mete_agua(L) y saca_agua(L), respectivamente para añadir o extraer una cantidad de agua (L) del depósito. Estas operaciones pueden ser invocadas por hilos concurrentes y se quiere asegurar un funcionamiento sincronizado, de manera que un hilo que trate de meter más agua de la que cabe en el depósito, deberá esperarse hasta que haya espacio suficiente. Análogamente, un hilo que intente sacar agua deberá esperar a que haya suficiente cantidad de agua.

```
void mete_agua (String hilo, float litros) {
… si el depósito se rebosa, esperar
… añadir el agua al depósito
… desbloquear si hay otros hilos esperando
}

void saca_agua (String hilo, float litros) {
… esperar hasta que haya agua suficiente
… retirar el agua del depósito
… desbloquear si hay otros hilos esperando
}
```

Implementar mediante semáforos las acciones de sincronización de estas dos operaciones. Introduce las variables auxiliares que consideres necesarias.

Posible ejecución del programa:

```
S-0 SACAR AGUA, agua actual=0.0 litros a sacar=26.0
        S-0 bloqueado hasta que haya agua en el deposito
M-0 METER AGUA: agua actual=0.0 litros a meter=14.0
        M-0 metiendo 14.0 de agua: agua actual=14.0
        S-0 bloqueado hasta que haya agua en el deposito
M-1 METER AGUA: agua actual=14.0 litros a meter=20.0
        M-1 metiendo 20.0 de agua: agua actual=34.0
M-2 METER AGUA: agua actual=34.0 litros a meter=17.0
        M-2 metiendo 17.0 de agua: agua actual=51.0
        S-0 sacando 26.0 litros de agua, quedan=25.0
M-3 METER AGUA: agua actual=25.0 litros a meter=17.0
        M-3 metiendo 17.0 de agua: agua actual=42.0
M-4 METER AGUA: agua actual=42.0 litros a meter=12.0
        M-4 metiendo 12.0 de agua: agua actual=54.0
M-5 METER AGUA: agua actual=54.0 litros a meter=12.0
        M-5 metiendo 12.0 de agua: agua actual=66.0
M-6 METER AGUA: agua actual=66.0 litros a meter=19.0
        M-6 metiendo 19.0 de agua: agua actual=85.0
M-7 METER AGUA: agua actual=85.0 litros a meter=24.0
        M-7 metiendo 24.0 de agua: agua actual=109.0
S-1 SACAR AGUA, agua actual=109.0 litros a sacar=8.0
        S-1 sacando 8.0 litros de agua, quedan=101.0
S-2 SACAR AGUA, agua actual=101.0 litros a sacar=11.0
        S-2 sacando 11.0 litros de agua, quedan=90.0
M-8 METER AGUA: agua actual=90.0 litros a meter=30.0
        M-8 metiendo 30.0 de agua: agua actual=120.0
M-9 METER AGUA: agua actual=120.0 litros a meter=25.0
        M-9 metiendo 25.0 de agua: agua actual=145.0
S-3 SACAR AGUA, agua actual=145.0 litros a sacar=27.0
        S-3 sacando 27.0 litros de agua, quedan=118.0
S-4 SACAR AGUA, agua actual=118.0 litros a sacar=6.0
        S-4 sacando 6.0 litros de agua, quedan=112.0
```

11. Un ascensor tiene una capacidad máxima de 6 personas y 450 kilogramos. Se quiere simular el comportamiento del ascensor mediante un conjunto de hilos que actúan como personas que entran y salen del aparato. Para ello, tienes que implementar dos operaciones, sube_persona(String nombre, float peso) y baja_persona(String nombre, float peso). Estas dos operaciones serán invocadas por los hilos que simulan las personas.

La operación de subir debe dejar bloqueada a la persona que hace la llamada mientras el ascensor esté lleno, o bien la carga en kilos no permita que la persona se monte en el ascensor. La operación de bajar debe actualizar el estado del ascensor y, si es el caso, inducir el desbloqueo de otras personas que puedan estar esperando para entrar.

Implementa mediante semáforos las acciones de sincronización de estas operaciones de subir y bajar. Introduce las variables auxiliares que consideres necesarias. *(2 puntos)*

```
void sube_persona (String nombre, float peso) {
        … esperar si no se puede subir
        … subir al ascensor
}
void baja_persona (String nombre, float peso) {
        … baja del ascensor
        … desbloquear
}
```

```
Claudia llama al ascensor, su peso es de 66.27Kg
        Claudia sube al ascensor
Maria llama al ascensor, su peso es de 61.8Kg
Sergio llama al ascensor, su peso es de 116.74Kg
Antonio llama al ascensor, su peso es de 107.05Kg
        Maria sube al ascensor
Jesus llama al ascensor, su peso es de 87.11Kg
        Jesus sube al ascensor
Ana llama al ascensor, su peso es de 52.69Kg
        Ana sube al ascensor
Andrea llama al ascensor, su peso es de 83.19Kg
        Sergio sube al ascensor
Antonio tiene que esperar, las condiciones actuales son [5, 384,61Kg]
Andrea tiene que esperar, las condiciones actuales son [5, 384,61Kg]
                ==>Claudia baja del ascensor. Las condiciones actuales son [4, 318,34Kg]
        Antonio sube al ascensor
                ==>Jesus baja del ascensor. Las condiciones actuales son [4, 338,28Kg]
        Andrea sube al ascensor
                ==>Sergio baja del ascensor. Las condiciones actuales son [4, 304,73Kg]
                ==>Maria baja del ascensor. Las condiciones actuales son [3, 242,93Kg]
                ==>Ana baja del ascensor. Las condiciones actuales son [2, 190,24Kg]
                ==>Antonio baja del ascensor. Las condiciones actuales son [1, 83,19Kg]
                ==>Andrea baja del ascensor. Las condiciones actuales son [0, ,00Kg]
```

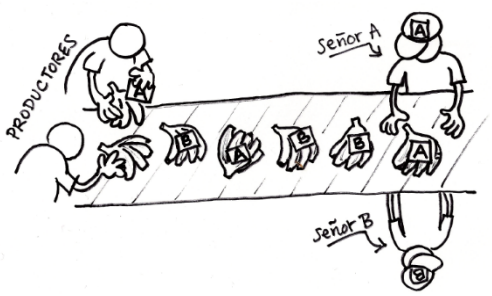
12. La urbanización "*Valle del Jerte*" dispone de una cancha de pádel para uso y disfrute de sus residentes. La política de uso de esta instalación deportiva es muy simple: a medida que van llegando jugadores a la cancha, estos van ocupando un puesto a la espera de completar el cupo de 4 jugadores, momento en el que pueden disputar un partido. Solo cuando llega el cuarto jugador, se disputa el partido. Si llegan jugadores mientras se está disputando un partido, estos deberán esperar a que la cancha quede libre.

Una vez el partido finaliza, los cuatro jugadores abandonan la cancha, dejándola libre, el último en abandonar la cancha avisa a los que están esperando para entrar a jugar.

Se pide escribir un algoritmo jugador de forma que se cumplan las reglas descritas empleando semáforos como herramienta de sincronización. Puede asumir que la acción de jugar un partido durará un tiempo finito e igual para todos los jugadores (10 sg) y además puede suponer que continuamente llegan jugadores a la cancha con la intención de jugar un partido. *(2 puntos)*

```
=>Antonio entra en la cancha
=>Maria entra en la cancha
=>Ana entra en la cancha
=>Jesus entra en la cancha
        Antonio está esperando a que lleguen más jugadores [1]
        Maria está esperando a que lleguen más jugadores [2]
        Ana está esperando a que lleguen más jugadores [3]
        Jesus: -¡¡Ya somos [4] podemos empezar el partido!!
                Jesus jugando partido
                Maria jugando partido
                Ana jugando partido
                Antonio jugando partido
                        Jesus abandonando cancha...
                        Maria abandonando cancha...
                        Ana abandonando cancha...
                        Antonio es el/la último/a en salir, avisa al resto
=>Claudia entra en la cancha
        Claudia está esperando a que lleguen más jugadores [1]
=>Andrea entra en la cancha
        Andrea está esperando a que lleguen más jugadores [2]
=>Sergio entra en la cancha
        Sergio está esperando a que lleguen más jugadores [3]
=>Esteban entra en la cancha
        Esteban: -¡¡Ya somos [4] podemos empezar el partido!!
                Esteban jugando partido
                Claudia jugando partido
                Andrea jugando partido
                Sergio jugando partido
                        Esteban abandonando cancha...
                        Claudia abandonando cancha...
                        Sergio abandonando cancha...
                        Andrea es el/la último/a en salir, avisa al resto
```

13. Estamos en el taller de empaquetado de una explotación platanera. Tenemos a varias personas (procesos productores) que colocan manojos de plátanos en una cinta transportadora. Los productores etiquetan las manos con una de estas dos calidades: A y B. En el otro extremo de la cinta hay dos personas (procesos consumidores) que recogen los plátanos de acuerdo con su calidad: una persona, el "señor A", recoge sólo los plátanos de calidad A y otra persona, el "señor B", los plátanos de calidad B. Los consumidores estarán trabajando mientras haya manojos de plátanos en la cinta.



Los plátanos van llegando por la cinta en el mismo orden en el que se van produciendo. Los plátanos se deben recoger también en orden de llegada: el primer manojo de plátanos de la cinta debe ser recogido por el consumidor encargado de la calidad de dicho manojo. El otro consumidor debe esperar hasta que el primer manojo de la cola sea de su calidad. Por ejemplo, si el primer manojo es de calidad A y el segundo de calidad B, el señor B debe esperarse hasta que el señor A recoja el primer mano de calidad A, cuando lo recoja el señor A, avisará al se ñor B.

Desarrolla una aplicación para la sincronización de los procesos productores y los consumidores, utilizando semáforos como herramienta de sincronización.
Asume que la cinta tiene tamaño ilimitado (no hay que controlar si se llena).

```
Productor, etiquetando manojo: platanos-B
-----------------------------------------
| platanos-B
-----------------------------------------

Productor, etiquetando manojo: platanos-A
-----------------------------------------
| platanos-A | platanos-B
-----------------------------------------

Productor, etiquetando manojo: platanos-B
-----------------------------------------
| platanos-B | platanos-A | platanos-B
-----------------------------------------

Productor, etiquetando manojo: platanos-A
-----------------------------------------
| platanos-A | platanos-B | platanos-A | platanos-B
-----------------------------------------

Productor, etiquetando manojo: platanos-A
-----------------------------------------
| platanos-A | platanos-A | platanos-B | platanos-A | platanos-B
-----------------------------------------

Señor B ==> recoge platanos
-----------------------------------------
| platanos-A | platanos-A | platanos-B | platanos-A
-----------------------------------------

Señor A ==> recoge platanos
-----------------------------------------
| platanos-A | platanos-A | platanos-B
-----------------------------------------

Señor B ==> recoge platanos
-----------------------------------------
| platanos-A | platanos-A
-----------------------------------------

Señor A ==> recoge platanos
-----------------------------------------
| platanos-A
-----------------------------------------

Señor A ==> recoge platanos
-----------------------------------------

-----------------------------------------
```