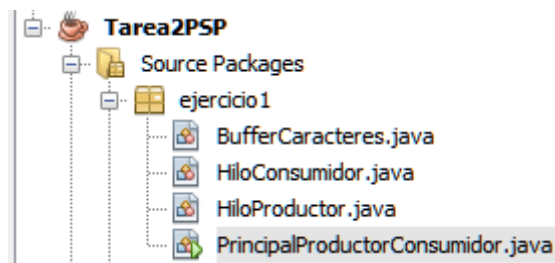


EJERCICIO 1

Ejercicio del tipo productor-consumidor en el que se nos pide que un hilo produzca hasta 15 caracteres dentro de un buffer compartido y otro hilo a su vez vaya recogiendo esos 15 caracteres vaciando el buffer.

Para la realización del mismo hemos creado 4 clases:



2 clases que son los hilos, una clase que contendrá el buffer junto con los métodos correspondientes de almacenamiento y otra clase con el método principal para la ejecución del programa

A continuación muestro de manera resumida(no se ve todo el código) que es lo que contiene y hace cada una de ellas:

- **Clase BufferCaracteres:** Contiene el vector de caracteres compartido entre los hilos, que iremos rellenando o vaciando de una manera síncrona mediante monitores en los 2 métodos que acceden al mismo recurso, en nuestro caso el buffer. Los métodos incrementar y decrementar hacen uso de los métodos wait() y notify() de la clase Thread para que haya una comunicación entre los hilos que acceden al recurso y así evitar inanición o interbloqueo entre los mismos.

```
public class BufferCaracteres {

    char[] bufferCaracteres = new char[6]; //Vector de 6 caracteres que hace de
    buffer
    public int indice;

    /**
     * Constructor de la clase que inicializa el valor de índice a 1
     */
    public BufferCaracteres() {
        indice = 0;
    }

    /**
     * Método monitor protegido para la concurrencia de datos compartidos entre
     * hilos que deposita un carácter aleatorio en el buffer de izquierda a
     * derecha mientras no esté lleno
     */
    public synchronized void incrementaBuffer() {

    }

    /**
     * Método monitor protegido para la concurrencia de datos compartidos entre
     * hilos que recoge un carácter aleatorio del buffer de derecha a izquierda
     * mientras no esté vacío
     */
    public synchronized void decrementaBuffer() {

    }
}
```

- **Clases HiloConsumidor e HiloProductor:** Al crear estas clases extendemos la clase Thread de esta manera heredaré de la misma y nos servirá para poder hacer los hilos de nuestra aplicación mediante la sobreescritura de la interfaz run(). Las clases son similares en su construcción la única diferencia es que una llama al método incrementar y la otra al método decrementar.

```
public class HiloConsumidor extends Thread {

    BufferCaracteres bc = new BufferCaracteres();

    public HiloConsumidor(BufferCaracteres bc) {
        this.bc = bc;
    }

    /**
     * Interfaz run() de la clase Thread en la que definimos la ejecución del
     * hilo. Llama 15 veces al método decrementaBuffer de la clase
     * BufferCaracteres
     */
    @Override
    public void run() {

        for (int i = 0; i < 15; i++) {
            bc.decrementaBuffer();
        }

    }

}

public class HiloProductor extends Thread {

    BufferCaracteres bc = new BufferCaracteres();

    public HiloProductor(BufferCaracteres bc) {
        this.bc = bc;
    }

    /**
     * Interfaz run() de la clase Thread en la que definimos la ejecución del
     * hilo. Llama 15 veces al método incrementaBuffer de la clase
     * BufferCaracteres
     */
    @Override
    public void run() {

        for (int i = 0; i < 15; i++) {
            bc.incrementaBuffer();
        }

    }

}
```

- **Clase PrincipalProductorConsumidor:** Contiene el método principal o main() de nuestra aplicación y en el vamos a instanciar la clase Buffer para luego pasarla como parámetros a los hilos productor y consumidor que creamos y posteriormente ejecutamos con el método star().

```
public class PrincipalProductorConsumidor {

    /**
     * En el método principal creamos una instancia de la clase BufferCaracteres
     * que luego pasaremos como parámetro tanto al hilo productor como al hilo
     * consumidor que vamos a instanciar a continuación. Ejecutamos los hilos
     * mediante el método star()
     *
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        BufferCaracteres bc = new BufferCaracteres();

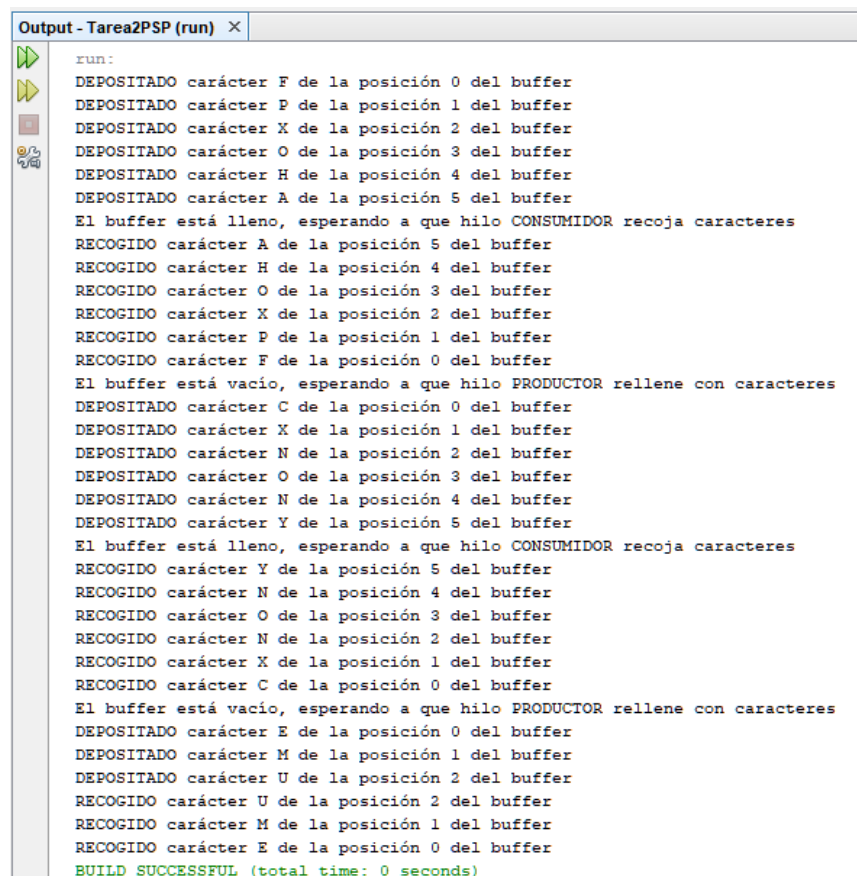
        HiloProductor productor = new HiloProductor(bc);
        productor.start();

        HiloConsumidor consumidor = new HiloConsumidor(bc);
        consumidor.start();

    }

}
```

Una vez tenemos conformadas nuestras clases procedemos a compilar y ejecutar el programa y este es el resultado que nos mostrará por pantalla:



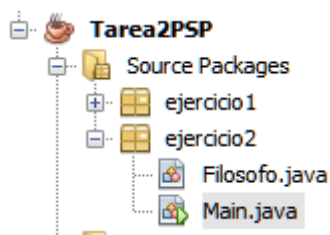
```
Output - Tarea2PSP (run) X
run:
DEPOSITADO carácter F de la posición 0 del buffer
DEPOSITADO carácter P de la posición 1 del buffer
DEPOSITADO carácter X de la posición 2 del buffer
DEPOSITADO carácter O de la posición 3 del buffer
DEPOSITADO carácter H de la posición 4 del buffer
DEPOSITADO carácter A de la posición 5 del buffer
El buffer está lleno, esperando a que hilo CONSUMIDOR recoja caracteres
RECOGIDO carácter A de la posición 5 del buffer
RECOGIDO carácter H de la posición 4 del buffer
RECOGIDO carácter O de la posición 3 del buffer
RECOGIDO carácter X de la posición 2 del buffer
RECOGIDO carácter P de la posición 1 del buffer
RECOGIDO carácter F de la posición 0 del buffer
El buffer está vacío, esperando a que hilo PRODUCTOR rellene con caracteres
DEPOSITADO carácter C de la posición 0 del buffer
DEPOSITADO carácter X de la posición 1 del buffer
DEPOSITADO carácter N de la posición 2 del buffer
DEPOSITADO carácter O de la posición 3 del buffer
DEPOSITADO carácter N de la posición 4 del buffer
DEPOSITADO carácter Y de la posición 5 del buffer
El buffer está lleno, esperando a que hilo CONSUMIDOR recoja caracteres
RECOGIDO carácter Y de la posición 5 del buffer
RECOGIDO carácter N de la posición 4 del buffer
RECOGIDO carácter O de la posición 3 del buffer
RECOGIDO carácter N de la posición 2 del buffer
RECOGIDO carácter X de la posición 1 del buffer
RECOGIDO carácter C de la posición 0 del buffer
El buffer está vacío, esperando a que hilo PRODUCTOR rellene con caracteres
DEPOSITADO carácter E de la posición 0 del buffer
DEPOSITADO carácter M de la posición 1 del buffer
DEPOSITADO carácter U de la posición 2 del buffer
RECOGIDO carácter U de la posición 2 del buffer
RECOGIDO carácter M de la posición 1 del buffer
RECOGIDO carácter E de la posición 0 del buffer
BUILD SUCCESSFUL (total time: 0 seconds)
```

Como podemos observar el hilo productor va rellendo el buffer de izquierda a derecha y una vez está completo se mantiene a la espera para que el hilo consumidor recoja de derecha a izquierda. Una vez consumido es el hilo consumidor el que espera para que el productor rellene. Así se repite hasta que se hayan depositado y recogido 15 caracteres.

EJERCICIO 2

En este ejercicio se nos pida que resolvamos el problema de “la cena de los filósofos” en el que 5 filósofos se pasan la vida pensando y comiendo. Para comer dispone cada uno de dos palillos a izquierda y derecha que tienen que compartir con los que tienen al lado, es decir no puede haber 2 filósofos contiguos que estén comiendo a la vez.

Para resolver este problema nos vamos a basar en el documento javadoc que nos proporciona la tarea que nos marca las clases, métodos y variables que podemos usar para la resolución del mismo:



Nuestra aplicación constará de 2 clases Filosofo donde se va a definir que es lo que tienen que hacer para sincronizarse entre ellos y la clase Main donde crearemos y ejecutaremos los 5 hilos, uno por cada filósofo

- **Clase Filosofo:** En esta clase, que hereda de la clase Thread, vamos a definir que es lo que tiene que hacer de manera general cada filósofo cuando se invoca a través de un hilo. Contiene un constructor al que le vamos a pasar como parámetros el índice o identificador de cada filósofo, el vector de la clase Semaphore uno por cada palillo que nos servirá para sincronizar hilos dando paso a un solo hilo por palillo hasta que este lo libere y lo pueda coger el siguiente filósofo y como tercer parámetro una matriz en la que cada fila definimos los palillos que necesita coger el filósofo para comer.

Esta clase contiene 3 métodos:

- pensar(): Muestra por pantalla que el filósofo está pensando y duerme el hilo 1 segundo para que sea más visual.
- comer(): Muestra por pantalla un mensaje de que el filósofo está hambriento e intenta conseguir los palillos que le hacen falta una vez los consigue muestra por pantalla que está comiendo, duerme el hilo 1 segundo y a continuación libera los palillos.
- run(): El enunciado del problema dice que los filósofos piensan y comen durante toda la vida, por lo tanto crearemos un bucle infinito en el que llamaremos a los métodos pensar() y comer() de manera indefinida.

```
public class Filosofo extends Thread {

    //Definimos las variables de la clase
    int miIndice;
    Semaphore[] semaforoPalillo;
    int[][] palillosFilosofo;

    public Filosofo(int miIndice, Semaphore[] semaforoPalillo, int[][]
palillosFilosofo) {

        this.miIndice = miIndice;
        this.semaforoPalillo = semaforoPalillo;
        this.palillosFilosofo = palillosFilosofo;
    }

    public void comer() {

        System.out.println("Filósofo " + (miIndice + 1) + " está HAMBRIENTO");
        //Trata de conseguir los palillos que necesita

        sleep(1000); //Cuando consigue los palillos come y duerme hilo 1 segundo

        System.out.println("Filósofo " + (miIndice + 1) + " ha TERMINADO DE COMER
y soltado los palillos " + (palillosFilosofo[miIndice][0] + 1) + " y " +
(palillosFilosofo[miIndice][1] + 1));
        //El filósofo libera los palillos después de haber comido

    }

    public void pensar() {

        System.out.println("Filósofo " + (miIndice + 1) + " está PENSANDO");
        sleep(1000); //Duerme el hilo durante 1 segundo

    }

    @Override
    public void run() {

        while (true) {
            pensar();
            comer();
        }
    }
}
```

- **Clase Main:** Dentro de esta clase principal que contiene el método main(), vamos a definir el vector de Semaphore con 5 instancias de Semaphore de un único permiso, también definimos la matriz de los palillos que necesita cada filósofo dependiendo de su índice, es decir cada fila corresponderá a los palillos que va a necesitar cada uno.

A continuación crearemos las 5 instancias de la clase filósofos como hilos y las ejecutamos.

```
public class Main {

    public static void main(String[] args) {

        int miIndice;
        //Creamos los 5 semáforos, uno por cada palillo, dando permiso de único
hilo
        Semaphore[] semaforoPalillo = new Semaphore[]{new Semaphore(1), new
Semaphore(1), new Semaphore(1), new Semaphore(1), new Semaphore(1)};
        //Definimos en una matriz los palillos que le corresponden a cada
filósofo
        int[][] palillosFilosofo = new int[][]{{0, 4}, {1, 0}, {2, 1}, {3, 2},
{4, 3}};

        //Creamos y ejecutamos los hilos correspondientes a los 5 filósofos
        miIndice = 0;
        Filosofo filosofo_1 = new Filosofo(miIndice, semaforoPalillo,
palillosFilosofo);
        filosofo_1.start();

        miIndice = 1;
        Filosofo filosofo_2 = new Filosofo(miIndice, semaforoPalillo,
palillosFilosofo);
        filosofo_2.start();

        miIndice = 2;
        Filosofo filosofo_3 = new Filosofo(miIndice, semaforoPalillo,
palillosFilosofo);
        filosofo_3.start();

        miIndice = 3;
        Filosofo filosofo_4 = new Filosofo(miIndice, semaforoPalillo,
palillosFilosofo);
        filosofo_4.start();

        miIndice = 4;
        Filosofo filosofo_5 = new Filosofo(miIndice, semaforoPalillo,
palillosFilosofo);
        filosofo_5.start();
    }
}
```

El resultado mostrado por pantalla cuando ejecutamos la aplicación es el siguiente:

```

Output - Tarea2PSP (run) x
run:
Filósofo 1 está PENSANDO
Filósofo 2 está PENSANDO
Filósofo 3 está PENSANDO
Filósofo 4 está PENSANDO
Filósofo 5 está PENSANDO
Filósofo 1 está HAMBRIENTO
Filósofo 2 está HAMBRIENTO
Filósofo 3 está HAMBRIENTO
Filósofo 4 está HAMBRIENTO
Filósofo 5 está HAMBRIENTO
Filósofo 1 está COMIENDO con los palillos 1 y 5
Filósofo 1 ha TERMINADO DE COMER y soltado los palillos 1 y 5
Filósofo 1 está PENSANDO
Filósofo 2 está COMIENDO con los palillos 2 y 1
Filósofo 2 ha TERMINADO DE COMER y soltado los palillos 2 y 1
Filósofo 1 está HAMBRIENTO
Filósofo 3 está COMIENDO con los palillos 3 y 2
Filósofo 2 está PENSANDO
Filósofo 3 ha TERMINADO DE COMER y soltado los palillos 3 y 2
Filósofo 3 está PENSANDO
Filósofo 4 está COMIENDO con los palillos 4 y 3
Filósofo 2 está HAMBRIENTO
Filósofo 3 está HAMBRIENTO
Filósofo 4 ha TERMINADO DE COMER y soltado los palillos 4 y 3
Filósofo 4 está PENSANDO
Filósofo 5 está COMIENDO con los palillos 5 y 4
Filósofo 4 está HAMBRIENTO
Filósofo 5 ha TERMINADO DE COMER y soltado los palillos 5 y 4
Filósofo 5 está PENSANDO
Filósofo 1 está COMIENDO con los palillos 1 y 5
Filósofo 5 está HAMBRIENTO
Filósofo 1 ha TERMINADO DE COMER y soltado los palillos 1 y 5
Filósofo 1 está PENSANDO
Filósofo 2 está COMIENDO con los palillos 2 y 1
Filósofo 1 está HAMBRIENTO
Filósofo 2 ha TERMINADO DE COMER y soltado los palillos 2 y 1
Filósofo 2 está PENSANDO
Filósofo 3 está COMIENDO con los palillos 3 y 2
Filósofo 2 está HAMBRIENTO
Filósofo 3 ha TERMINADO DE COMER y soltado los palillos 3 y 2
Filósofo 4 está COMIENDO con los palillos 4 y 3
Filósofo 3 está PENSANDO
Filósofo 4 ha TERMINADO DE COMER y soltado los palillos 4 y 3
Filósofo 4 está PENSANDO
Filósofo 5 está COMIENDO con los palillos 5 y 4
Filósofo 3 está HAMBRIENTO
Filósofo 4 está HAMBRIENTO
Filósofo 5 ha TERMINADO DE COMER y soltado los palillos 5 y 4
Filósofo 5 está PENSANDO
Filósofo 1 está COMIENDO con los palillos 1 y 5
BUILD STOPPED (total time: 11 seconds)

```

Podemos ver que todos los filósofos comienzan pensando, luego pasan a estar hambrientos y a partir de aquí ya comienza la “lucha” por los palillos y el control de los hilos pasa a manos de la clase Semaphore que es quien gestiona la concurrencia y sincronización evitando que se produzca interbloqueo o “deadlock”.

Observamos que ningún filósofo contiguo está comiendo a la vez porque no puede y también vemos que todos los filósofos más tarde o más temprano comen y vuelven a pensar y a estar hambrientos. Sería así un número indefinido de veces si no fuera porque he parado la ejecución para poder recoger el pantallazo.