MEDAC
Instituto Oficial de Formación Profesional

# Servicios y Procesos Act 6

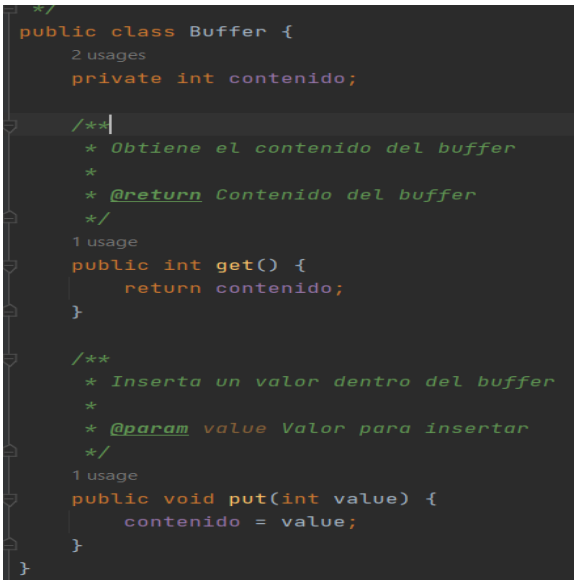
Mathias Ferreira Ferreira

Miguel Ángel Fernández

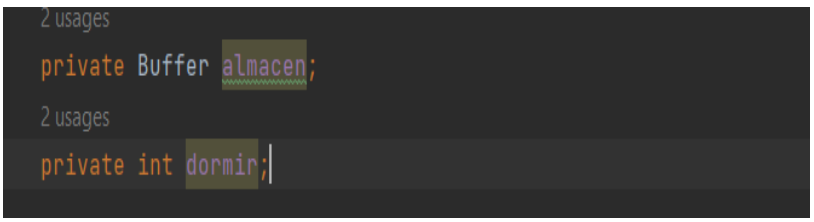
Iker Ruiz

## 6.2. Redactar un pdf con el resumen de lo que hace el código.

El código se divide en 3 clases , la primera un buffer ,la segunda la clase consumidor y la última la clase productor ,la clase buffer se encarga de almacenar el contenido(el valor a consumir) donde almacena el valor que va poniendo el productor mediante su método “put” y nos devuelve el valor de producción.




La siguiente clase productor instancia la clase buffer donde almacenará los valores que producirá y se declara una variable de tipo int donde se le indicará el tiempo a esperar del a producir.




Creamos un constructor de tipo productor que le pase los parámetros anteriores esta clase extiende de Thread eso quiere decir que debe declarar los método de ello.Este método es ‘run()’ esta debe ser sobrecargados.Dentro de ella creamos un for que recorra 10 vueltas donde dentro de ese for pondremos el valor de cada vuelta si da 2 vueltas meterá 2 valores.Esto se almacena en el buffer donde guarda ese valor como anteriormente hemos dicho con ‘almacén.put(valor)’

```java
@Override
public void run() {
    for (int i = 0; i < 10; i++) {
        almacen.put(i);
        System.out.println("Productor pone: " + i);
```

Aquí se puede observar lo dicho anteriormente.Posteriormente a ello se encargará de crear un ‘sleeper’ que como anteriormente hemos creado una variable dormir cual esperará el tiempo indicado que pongamos esta debe ser inferior al tiempo del consumidor por que si produce más lento que el tiempo de consumo estaría fuera de ‘stock ’o sin existencia.

```java
try {
    sleep(dormir);
} catch (InterruptedException e) {
    System.err.println("Error en el productor: " + e.toString());
}
```

(AQUI UNA CAPTURA DE COMO DEBERÍA SER)

Ahora vamos a explicar el comportamiento de consumidor, en su constructor también le pasaremos el buffer donde se almacena la información que almacena,y el tiempo de espera que debe hacer

```
2 usages
private Buffer almacen;
2 usages
private int dormir;

/**
 * Constructor del consumidor
 * @param almacen Buffer de donde se obtendrán los recursos
 * @param dormir Tiempo que dormirá el consumidor
 */
2 usages
public Consumidor(Buffer almacen, int dormir){
    this.almacen = almacen;
    this.dormir = dormir;
}
```

Esta clase al ser implementada por 'THREAD' deberá implementar sus métodos y en este caso será 'run()' que en este caso también recorrerá 10 vueltas igual que lo que se produce para ser consumido aquí al recorrer obtenemos el primer contenido o valor de buffer que es almacenado ese valor por productor

```
@Override
public void run()
{
    int valor = 0;
    for (int i = 0; i < 10; i++)
    {
        valor = almacen.get();
        System.out.println("Consumidor saca: "+ valor);
```

dentro del for tenemos que encargarnos de pasarle la el valor dormir cual debe ser superior al valor de productor por que si consume más rápido que lo que produce no habrá 'stock' esta le pasara un tiempo superior.

```
try
{
    sleep(dormir);
}
catch (InterruptedException e)
{
    System.err.println("Error en el consumidor: " + e.toString());
}
```

Se captura el problema con try/catch.

Y para comprobar los hechos anteriores se le pasara una constante con valores de sleep tanto como para consumidor como para productor se le para el buffer (que sería el almacén) donde guardará los valores e instanciamos nuestras clase productor y consumidor

```
public static void main(String[] args)
{
    final int DORMIR_PRODUCTOR = 1000, DORMIR_CONSUMIDOR = 2000;

    Buffer almacen = new Buffer();
    Productor productor = new Productor(almacen, DORMIR_PRODUCTOR);
    Consumidor consumidor = new Consumidor(almacen, DORMIR_CONSUMIDOR);

    productor.start();
    consumidor.start();
```

La velocidad de producción debe ser menor a la velocidad de consumición.

Para el Ejemplo del video numero 2 tenemos las mismas clases pero su funcionamiento es diferente →

En la clase buffer tenemos los mismo métodos get y put pero la diferencia son la siguiente :

En el método 'put()' tenemos una condición donde se controla si hay disponibilidad o no de stock en el almacén donde si no dentro del buffer no está disponible está esperará , en el otro

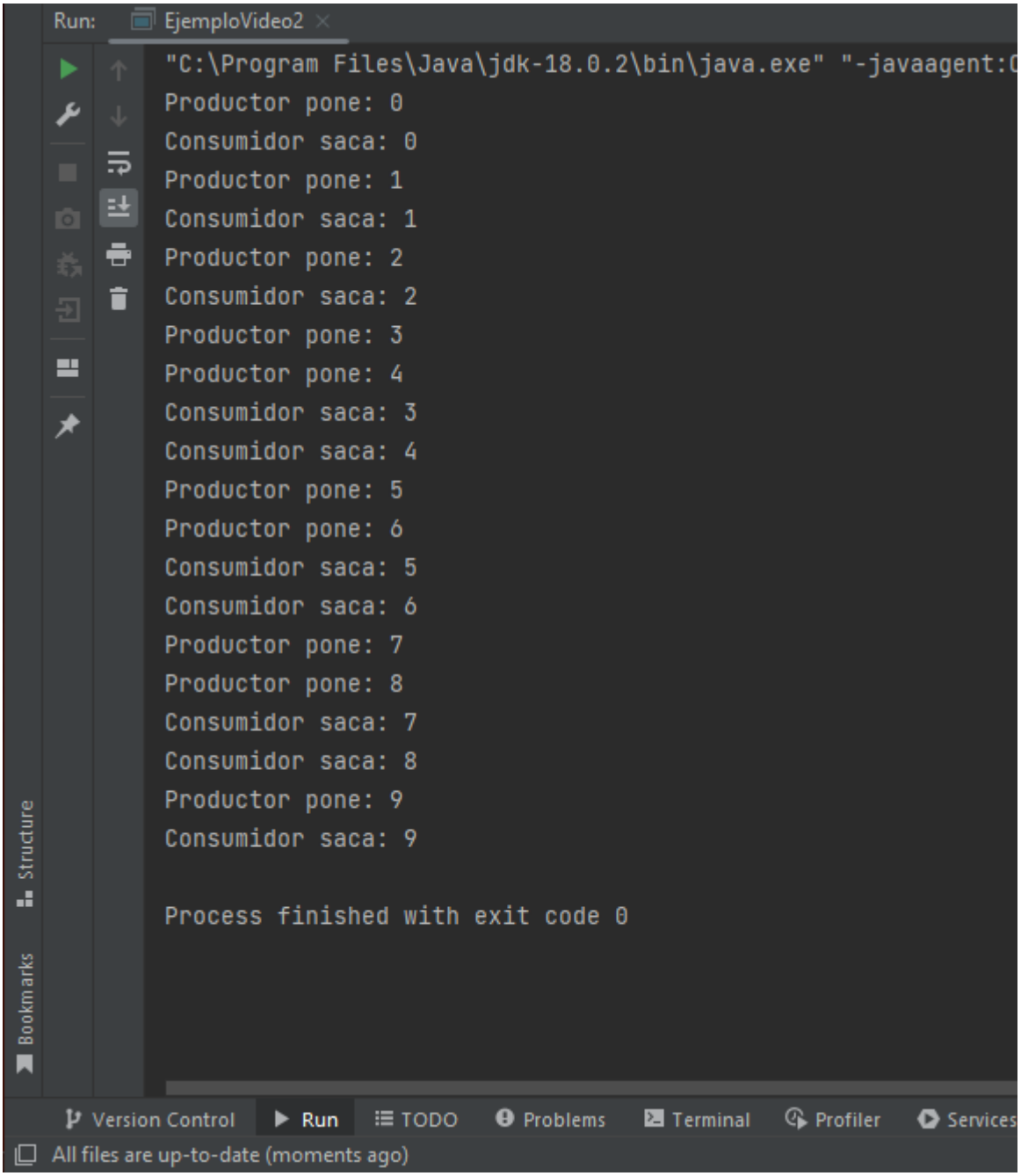
caso no controla si tiene o no disponibilidad.Si esta disponible esta lo notificará y devolverá el contenido del almacén del buffer y luego nos pasara false disponible por que será consumido.

```
1 usage
public synchronized int get() {
    // Mientras el buffer no esté disponible
    while (disponible == false)
    {
        try
        {
            // me espero a que produzcan
            wait();
        }
        catch (InterruptedException e) {}
    }
    // Cuando vuelve a estar disponible, notifico que está disponible
    disponible = false;
    notify();
    return contenido;
}
```

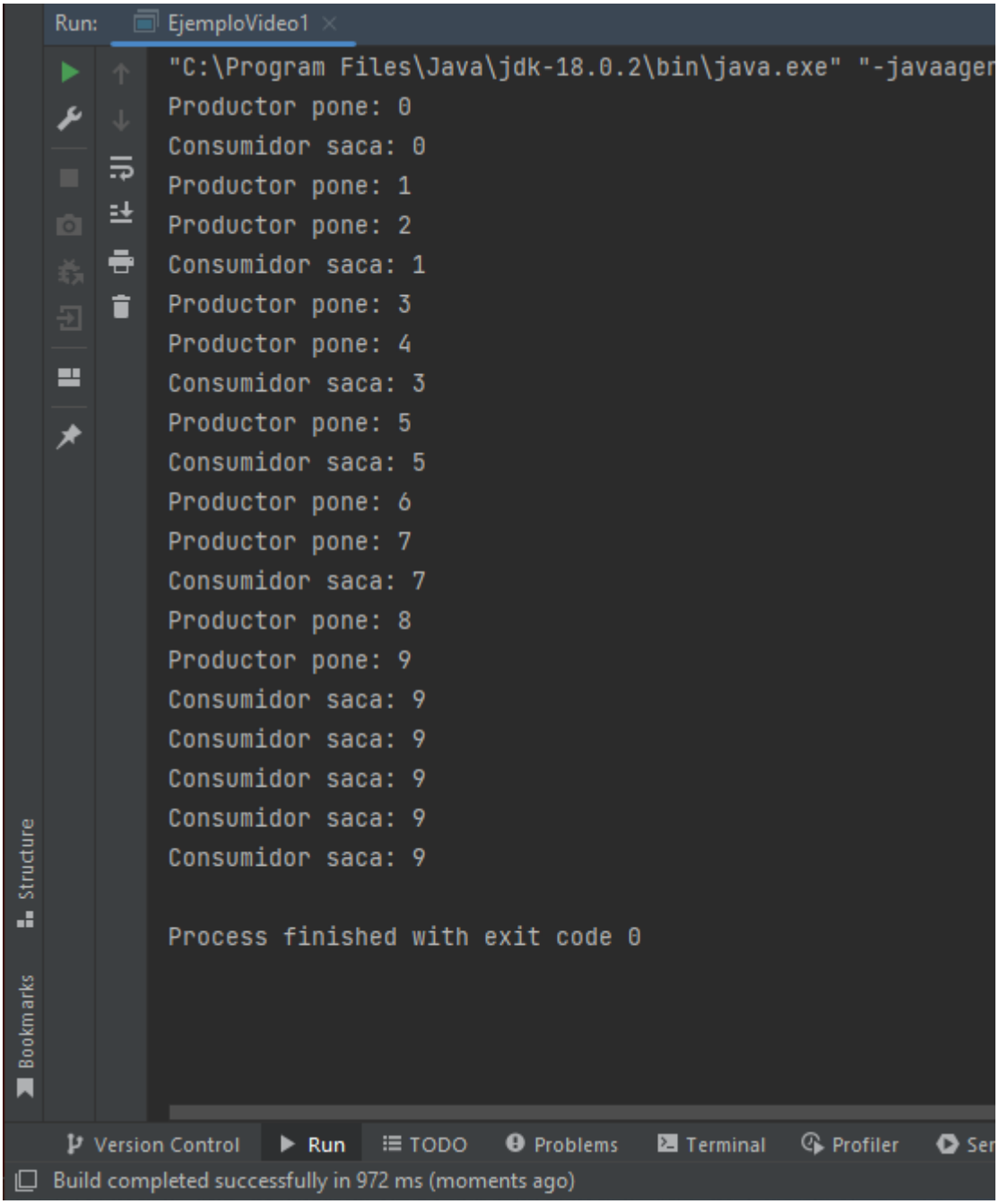
en el método ‘put()’ de esta clase si el buffer está libre o disponible esta esperará a que sea consumido por qué hay stock en el almacén cuando este libre se pondrá el valor dentro del contenido y pondrá disponible para ser consumido lo cual lo notificará.

```
    public synchronized void put(int value)
    {
        // Mientras el buffer esté disponible
        while (disponible == true)
        {
            try
            {
                // me espero a que consuman
                wait();
            }
            catch (InterruptedException e) {}
        }
        // Cuando vuelve a estar disponible, notifico que está disponible
        contenido = value;
        disponible = true;
        notify();
    }
}
```

En conclusión el ejemplo 1 no controla la entrada y salida de almacén y el ejemplo 2 controla la salida y entrada del almacén notificando la situación.




(Este código controlando la entrada de producción y consumición )




(Aquí al no controlar si ha acabado el stock al producir más rápido el consumidor sacar el valor que hay dentro del buffer varias veces hasta que termine el for).

## 6.3. Extrapolar el caso a un productor y varios consumidores.

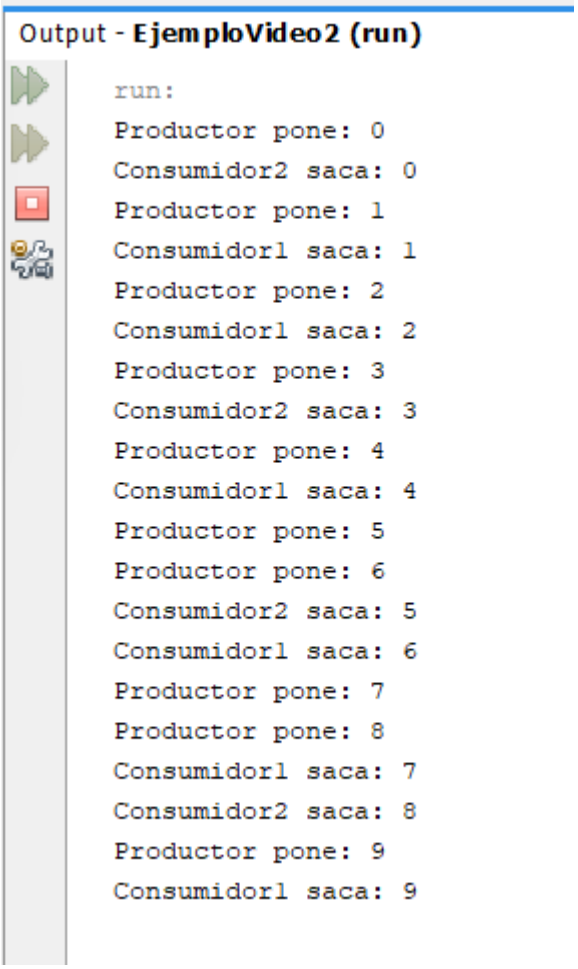
Lo primero que hemos realizado ha sido añadirle un identificador a los consumidores para saber diferenciarlos

```
    private int identificador;

    /**
     * Constructor del consumidor
     * @param almacen Buffer de donde se obtendrán los recursos
     * @param dormir Tiempo que dormirá el consumidor
     */
    public Consumidor(Buffer almacen, int dormir,int identificador)
    {
        this.almacen = almacen;
        this.dormir = dormir;
        this.identificador=identificador;
    }

    public void run()
    {
        int valor = 0;
        for (int i = 0; i < 10; i++)
        {
            valor = almacen.get();
            System.out.println("Consumidor" + identificador + " saca: "+ valor);
            try
            {
                sleep(dormir);
            }
            catch (InterruptedException e)
            {
                System.err.println("Error en el consumidor: " + e.toString());
            }
        }
    }
}
```

A continuación en el main creamos dos consumidores con tiempos de espera distintos y un productor y luego ejecutamos.

```
public class EjemploVideo2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        final int DORMIR_PRODUCTOR = 1000, DORMIR_CONSUMIDOR = 2000;

        Buffer almacen = new Buffer();
        Productor productor = new Productor(almacen, DORMIR_PRODUCTOR);
        Consumidor consumidor = new Consumidor(almacen, DORMIR_CONSUMIDOR,1);
        Consumidor consumidor2 = new Consumidor(almacen, DORMIR_CONSUMIDOR+1030,2);

        productor.start();
        consumidor.start();
        consumidor2.start();
    }

}
```

Ejecución:




## 6.4. Realizar el mismo ejercicio pero ahora haciendo uso de la clase Semaphore.

La clase semáforo lo que hace es ejecutar el proceso de forma sincronizada y en orden que controla la entrada y salida de productor y consumidor.

```java
package ejemplovideo2;
import java.util.concurrent.Semaphore;

public class semaforo {
    private int item;

    private Semaphore semaforoconsumer = new Semaphore(0);
    private Semaphore semaforoproducer = new Semaphore(1);

    public void get(){
        try{
            semaforoconsumer.acquire();
        }catch(InterruptedException e){
            System.out.println("ERROR: " + e);
        }
        System.out.println("El consumidor ha consumido: " + item);
        semaforoproducer.release();
    }
    public void put(int item){
        try{
            semaforoproducer.acquire();
        }catch(InterruptedException e){
            System.out.println("ERROR: " + e);
        }
        this.item = item;
        System.out.println("El productor ha producido: " + item);
        semaforoconsumer.release();
    }
}
```