Tarea PSP02

Yanet López Rodríguez
19/01/2024

Indice

Introducción

En esta actividad deberemos realizar los siguientes dos ejercicios:

💛**Ejercicio 1**
De igual manera a lo visto en el tema, ahora te proponemos un ejercicio del tipo productor-consumidor que mediante un hilo productor almacene datos (15 caracteres) en un búfer compartido, de donde los debe recoger un hilo consumidor (consume 15 caracteres). La capacidad del búfer ahora es de 6 caracteres, de manera que el consumidor podrá estar cogiendo caracteres del búfer siempre que éste no esté vacío. El productor sólo podrá poner caracteres en el búfer, cuando esté vacío o haya espacio.

💛**Ejercicio 2**
De igual manera a lo visto en el tema, ahora te proponemos que resuelvas el clásico problema denominado "La cena de los filósofos" utilizando la clase Semaphore del paquete java.util.concurrent.

El problema es el siguiente: Cinco filósofos se sientan alrededor de una mesa y pasan su vida comiendo y pensando. Cada filósofo tiene un plato de arroz chino y un palillo a la izquierda de su plato. Cuando un filósofo quiere comer arroz, cogerá los dos palillos de cada lado del plato y comerá. El problema es el siguiente: establecer un ritual (algoritmo) que permita comer a los filósofos. El algoritmo debe satisfacer la exclusión mutua (dos filósofos no pueden emplear el mismo palillo a la vez), además de evitar el interbloqueo y la inanición.

En este archivo haremos un manual en el que explicaremos cada código y los probaremos.

💛 Ejercicio 1

De igual manera a lo visto en el tema, ahora te proponemos un ejercicio del tipo productor-consumidor que mediante un hilo productor almacene datos (15 caracteres) en un búfer compartido, de donde los debe recoger un hilo consumidor (consume 15 caracteres). La capacidad del búfer ahora es de 6 caracteres, de manera que el consumidor podrá estar cogiendo caracteres del búfer siempre que éste no esté vacío. El productor sólo podrá poner caracteres en el búfer, cuando esté vacío o haya espacio.

Para realizar esta actividad deberemos crear cuatro clases “Main”, “Buffer”, “Consumidor” y “Productor”. Primero crearemos la clase “Buffer”.

En la clase “B*uffer*” lo primero que haremos será declarar las variables y crear el constructor.

```
public class Buffer {
    //DECLARAMOS LAS VARIABLES
    private char[] buffer;
    private int sucesivo;
    private boolean completo, libre;

    //CREAMOS EL CONSTRUCTOR
    public Buffer(int longitud){
        this.buffer = new char[longitud];
        this.sucesivo = 0;
        this.completo = false;
        this.libre = true;
    }
```

Ahora creamos el método consumir, este tendrá un bucle “*while*” que mientras que el “*buffer*” tenga espacio libre, hará que el hilo actual espere hasta que se reciba una notificación.

Restamos 1 a la variable “*sucesivo”*, pasamos la variable “*completo”* a false y si “*sucesivo”* es igual a cero pasamos la variable “*libre”* a verdadero, notificamos a los hilos en espera que pueden reanudar su ejecución y por ultimo devolvemos el valor de la posición de “*sucesivo”* del “*buffer*”.

En resumen, este método es utilizado por hilos para consumir un elemento del “*buffer*” en un entorno de concurrencia, garantizando que solo un hilo pueda consumir a la vez y evitando condiciones de carrera.

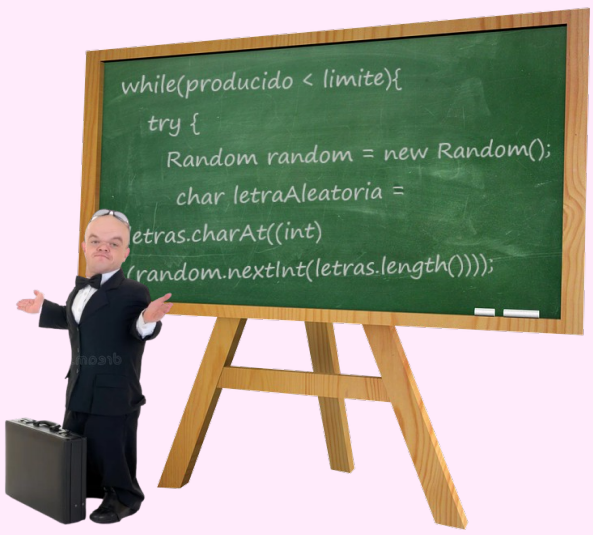
```
//CREAMOS EL MÉTODO consumir
public synchronized char consumir(){
    //MIENTRAS QUE EL BUFFER TENGA ESPACIO LIBRE
    while (this.libre) {
        try {
            //HACEMOS QUE EL HILO ACTUAL ESPERE HASTA QUE SE RECIBA UNA NOTIFICACIÓN
            wait();
        //SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
        } catch (InterruptedException ex) {
            Logger.getLogger(Buffer.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    //RESTAMOS 1 A LA VARIABLE sucesivo
    this.sucesivo--;
    //PASAMOS LA VARIABLE completo A FALSE
    this.completo = false;

    //SI sucesivo ES IGUAL A CERO PASAMOS LA VARIABLE LIBRE A VERDADERO
    if (this.sucesivo == 0) {
        this.libre = true;
    }

    //NOTOFOCAMOS A LOS HILOS EN ESPERA QUE PUEDEN REANUDAR SU EJECUCION
    notifyAll();

    //DEVOLVEMOS EL VALOR DE LA POSICIÓN DE sucesivo DEL buffer
    return this.buffer[this.sucesivo];
}
```

Y por último creamos el método producir, este tendrá un bucle “*while*” que mientras que el “*buffer*” no tenga espacio libre, hará que el hilo actual espere hasta que se reciba una notificación.

Guardamos en la posición “*sucesivo*” de “*buffer*” el valor de “*letra*”, le sumamos 1 a la variable “*sucesivo*”, pasamos la variable “*libre*” a falso y si “*sucesivo*” vale lo mismo que el largo de “*buffer*” pasamos la variable “*completo*” a verdadero. Por último notificamos a los hilos en espera que pueden reanudar su ejecución.

En resumen, este método permite agregar un carácter al buffer, siempre y cuando haya espacio disponible. Si no hay espacio disponible, el hilo que intenta producir debe esperar hasta que haya espacio.

```
    //CREAMOS EL MÉTODO producir
    public synchronized void producir(char letra){
        //MIENTRAS QUE EL BUFFER NO TENGA ESPACIO LIBRE
        while(this.completo){
            try {
                //HACEMOS QUE EL HILO ACTUAL ESPERE HASTA QUE SE RECIBA UNA NOTIFICACIÓN
                wait();
                //SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
            } catch (InterruptedException ex) {
                Logger.getLogger(Buffer.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

        //GUARDAMOS EN LA POSICIÓN sucesivo DE buffer EL VALOR DE letra
        this.buffer[this.sucesivo] = letra;

        //LE SUMAMOS 1 A LA VARIABLE sucesivo
        this.sucesivo++;
        //PASAMOS LA VARIABLE libre A FALSO
        this.libre = false;

        //SI sucesivo VALE LO MISMO QUE EL LARGO DE buffer
        if (this.sucesivo == this.buffer.length) {
            //PASAMOS LA VARIABLE completo A VERDADERO
            this.completo = true;
        }

        //NOTOFOCAMOS A LOS HILOS EN ESPERA QUE PUEDEN REANUDAR SU EJECUCION
        notifyAll();
    }
}
```



La segunda clase que crearemos será la clase “Productor”(esta clase extenderá de “*Thread*”). Lo primero que haremos será declarar las variables y crear el constructor, este recibirá como parámetro un objeto “B*uffer*”.

```
public class Productor extends Thread{

    //DECLARAMOS LAS VARIABLES
    private Buffer buffer;
    private final String letras = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private int producido;
    private final int limite = 15;

    //CREAMOS EL CONSTRUCTOR EL CUAL RECIBIRÁ COMO PARAMETRO UN OBJETO Buffer
    public Productor (Buffer buffer){
        this.producido = 0;
        this.buffer = buffer;
    }
```

Ahora creamos el método “*run*”, este tendrá un bucle “*while*” que se ejecutará mientras “*producido”* sea menor a “*limite*”, dentro de este instanciamos la clase “*Random*”, generamos un número aleatorio para escoger una letra de nuestro String “*letras*”, llamamos a la función “*producir*” de la clase “*Buffer*” y le pasamos la letra aleatoria como parámetro.

Le sumamos 1 a la variable “*producido*”, mostramos por pantalla la letra depositada en el “*buffer*” y por último hacemos una pausa, la cual puede durar de 0 a 6000 milisegundos.

En resumen, este código simula un productor que deposita letras aleatorias en un “*buffer”* y hace pausas entre cada depósito.

```
//CREAMOS EL METODO RUN
public void run(){
    //CREAMOS UN BUCLE while QUE SE EJECUTARÁ MIENTRAS producido SEA MENOR A limite
    while(producido < limite){
        try {
            //INSTANCIAMOS LA CLASE Random
            Random random = new Random();
            //GENERAMOS UN NÚMERO ALEATORIO PARA ESCOGER UNA LETRA DE NUESTRO String
            char letraAleatoria = letras.charAt((int)(random.nextInt(letras.length())));
            //LLAMAMOS A LA FUNCIÓN producir DE LA CLASE Buffer Y LE PASAMOS LA LETRA ALEATORIA COMO PARÁMETRO
            buffer.producir(letraAleatoria);
            //LE SUMAMOS 1 A LA VARIABLE producido
            producido++;
            //MOSTRAMOS POR PANTALLA LA LETRA DEPOSITADA EN EL BUFFER
            System.out.println("Depositando el cáracter " + letraAleatoria + " en el buffer");
            //HACEMOS UNA PAUSA QUE PUEDE DURAR DE 0 A 6000 MILISEGUNDOS
            sleep((long)(random.nextInt(6000)));
        //SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
        } catch (InterruptedException ex) {
            Logger.getLogger(Productor.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
}
```

La tercera clase que crearemos será la clase "*Consumidor*" (esta clase extenderá de "*Thread*"). Lo primero que haremos será declarar las variables y crear el constructor, este recibirá como parámetro un objeto "B*uffer*".

```
public class Consumidor extends Thread{

    //DECLARAMOS LAS VARIABLES
    private Buffer buffer;
    private int consumido;
    private final int limite = 15;

    //CREAMOS EL CONSTRUCTOR
    public Consumidor (Buffer buffer){
        this.consumido = 0;
        this.buffer = buffer;
    }
```

Ahora creamos el método "*run*", este tendrá un bucle "*while*" que se ejecutará mientras "*consumido*" sea menor a "*limite*", dentro de este obtenemos la ultima letra del "*buffer*" y la almacenamos en la variable "*letra*", le sumamos 1 a la variable "*consumido*", mostramos por pantalla la letra que se recoge y por último hacemos una pausa que puede durar de 0 a 5000 milisegundos.

En resumen, este método representa el comportamiento de un consumidor que recoge letras del "*buffer*" y las muestra por pantalla, mientras la cantidad de letras recogidas no supere un límite determinado.

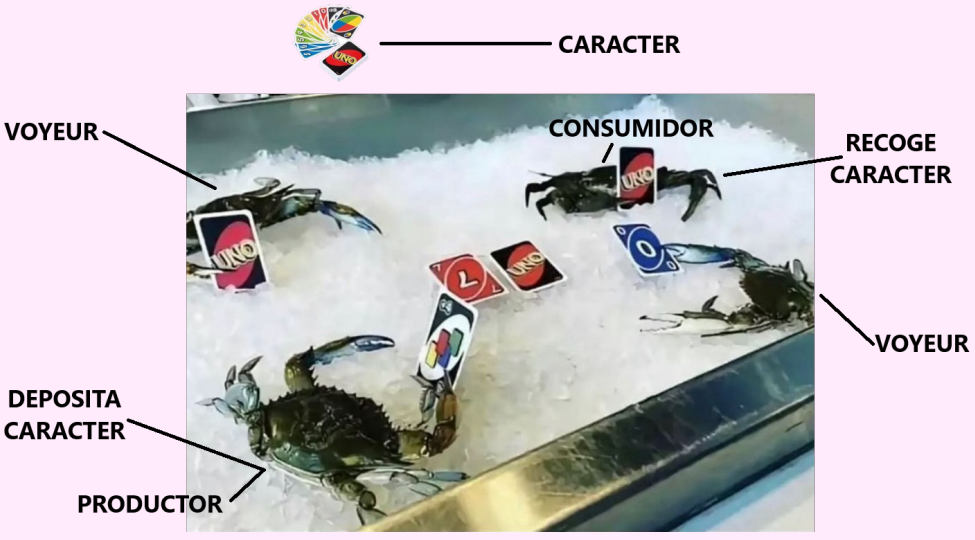
```
    //CREAMOS EL MÉTODO run
    public void run(){
        //CREAMOS UN BUCLE while QUE SE EJECUTARÁ MIENTRAS consumido SEA MENOR A limite
        while(consumido < limite){
            try {
                //OBTENEMOS LA ULTIMA LETRA DEL BUFFER Y LA ALMACENAMOS EN LA VARIABLE letra
                char letra = buffer.consumir();
                //LE SUMAMOS 1 A LA VARIABLE consumido
                consumido++;
                //MOSTRAMOS POR PANTALLA LA LETRA QUE SE RECOGE
                System.out.println("Recogiendo el carácter " + letra + " del buffer");
                //HACEMOS UNA PAUSA QUE PUEDE DURAR DE 0 A 5000 MILISEGUNDOS
                sleep((long)(Math.random() * 5000));
            //SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
            } catch (InterruptedException ex) {
                Logger.getLogger(Productor.class.getName()).log(Level.SEVERE, null, ex);
            }

        }
    }
}
```

Por último creamos la clase Main. Primero debemos crear una variable con la capacidad del "*buffer*", creamos tres instancias, una de la clase "*buffer*" pasándole la capacidad como argumento, otra de la clase "*productor*" pasándole "*buffer*" como argumento y por último una de la clase "*consumidor*" pasándole "*buffer*" como argumento.

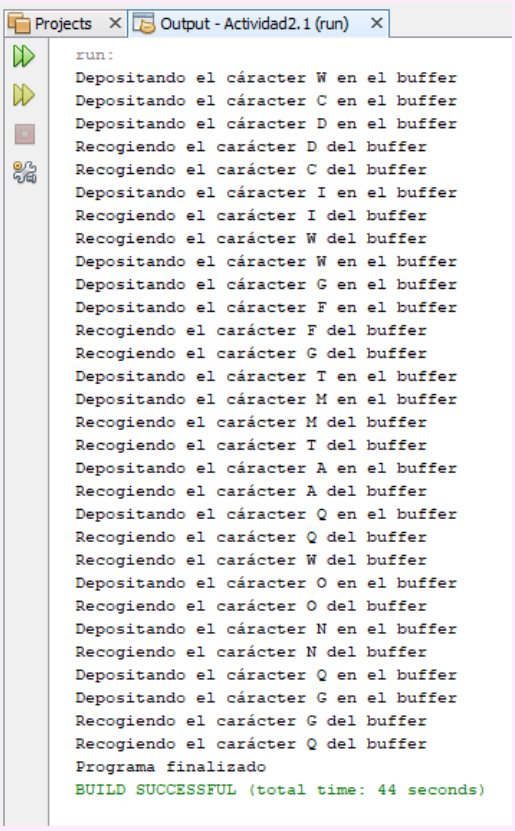
Llamamos al método "*start*" de la clase "*Productor*", pausamos la ejecución durante 5 segundos, llamamos al método "*start*" de la clase "*Consumidor*", esperamos a que el hilo de "*Productor*" termine antes de continuar y luego esperamos por el hilo de "*Consumidor*". Y por último mostramos por pantalla que el programa ha finalizado.

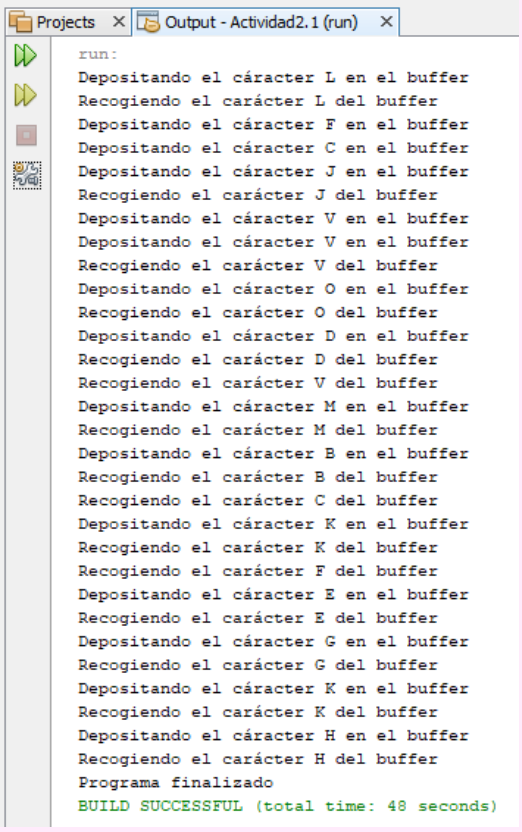
En resumen, este código crea un buffer con una capacidad específica, crea un productor y un consumidor que operan sobre ese buffer, y luego inicia la ejecución de ambos.

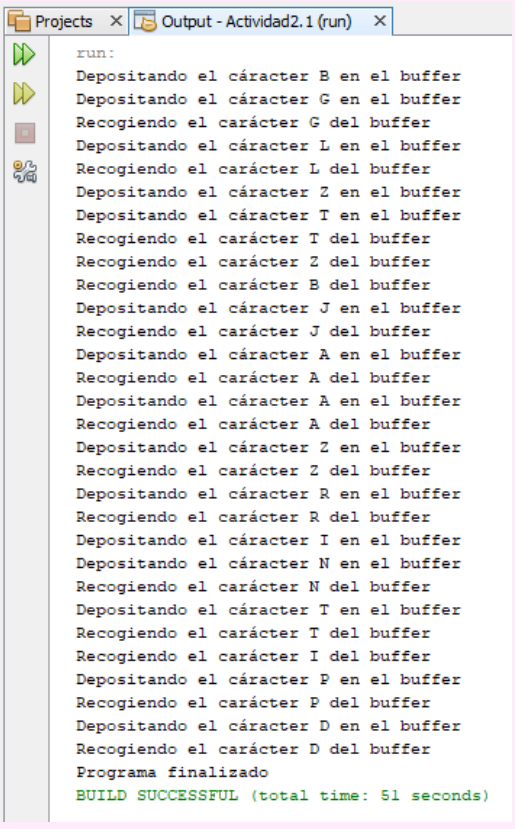
```java
public class Main {
    public static void main(String[] args){
        //CREAMOS UNA VARIABLE CON LA CAPACIDAD DEL BUFFER
        int capacidad = 6;

        try {
            //CREAMOS UNA INSTANCIA DE LA CLASE Buffer PASÁNDOLE LA CAPACIDAD COMO ARGUMENTO
            Buffer buffer = new Buffer(capacidad);
            //CREAMOS UNA INSTANCIA DE LA CLASE Productor PASÁNDOLE buffer COMO ARGUMENTO
            Productor productor = new Productor(buffer);
            //CREAMOS UNA INSTANCIA DE LA CLASE Consumidor PASÁNDOLE buffer COMO ARGUMENTO
            Consumidor consumidor = new Consumidor(buffer);

            //LLAMAMOS AL MÉTODO start DE LA CLASE Productor
            productor.start();
            //PAUSAMOS LA EJECUCIÓN DURANTE 5 SEGUNDOS
            Thread.sleep(5000);
            //LLAMAMOS AL MÉTODO start DE LA CLASE Consumidor
            consumidor.start();

            //ESPERAMOS A QUE EL HILO DE Productor TERMINE ANTES DE CONTINUAR
            productor.join();
            //ESPERAMOS A QUE EL HILO DE Consumidor TERMINE ANTES DE CONTINUAR
            consumidor.join();
            //MOSTRAMOS POR PANTALLA QUE EL PROGRAMA HA FINALIZADO
            System.out.println("Programa finalizado");

        //SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
        } catch (InterruptedException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

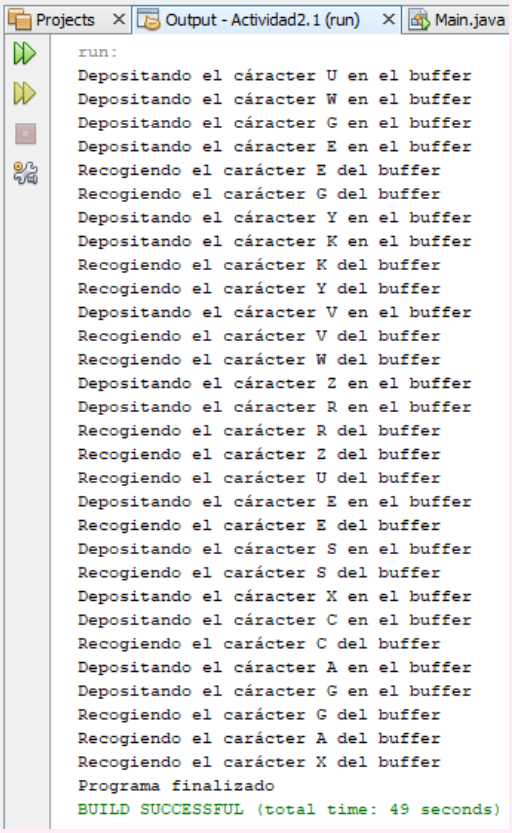


Pruebas


Projects
Output - Actividad2.1 (run)
run:
Depositando el cáracter W en el buffer
Depositando el cáracter C en el buffer
Depositando el cáracter D en el buffer
Recogiendo el carácter D del buffer
Recogiendo el carácter C del buffer
Depositando el cáracter I en el buffer
Recogiendo el carácter I del buffer
Recogiendo el carácter W del buffer
Depositando el cáracter W en el buffer
Depositando el cáracter G en el buffer
Depositando el cáracter F en el buffer
Recogiendo el carácter F del buffer
Recogiendo el carácter G del buffer
Depositando el cáracter T en el buffer
Depositando el cáracter M en el buffer
Recogiendo el carácter M del buffer
Recogiendo el carácter T del buffer
Depositando el cáracter A en el buffer
Recogiendo el carácter A del buffer
Depositando el cáracter Q en el buffer
Recogiendo el carácter Q del buffer
Recogiendo el carácter W del buffer
Depositando el cáracter O en el buffer
Recogiendo el carácter O del buffer
Depositando el cáracter N en el buffer
Recogiendo el carácter N del buffer
Depositando el cáracter Q en el buffer
Depositando el cáracter G en el buffer
Recogiendo el carácter G del buffer
Recogiendo el carácter Q del buffer
Programa finalizado
BUILD SUCCESSFUL (total time: 44 seconds)


Projects
Output - Actividad2.1 (run)
run:
Depositando el cáracter L en el buffer
Recogiendo el carácter L del buffer
Depositando el cáracter F en el buffer
Depositando el cáracter C en el buffer
Depositando el cáracter J en el buffer
Recogiendo el carácter J del buffer
Depositando el cáracter V en el buffer
Depositando el cáracter V en el buffer
Recogiendo el carácter V del buffer
Depositando el cáracter O en el buffer
Recogiendo el carácter O del buffer
Depositando el cáracter D en el buffer
Recogiendo el carácter D del buffer
Recogiendo el carácter V del buffer
Depositando el cáracter M en el buffer
Recogiendo el carácter M del buffer
Depositando el cáracter B en el buffer
Recogiendo el carácter B del buffer
Recogiendo el carácter C del buffer
Depositando el cáracter K en el buffer
Recogiendo el carácter K del buffer
Recogiendo el carácter F del buffer
Depositando el cáracter E en el buffer
Recogiendo el carácter E del buffer
Depositando el cáracter G en el buffer
Recogiendo el carácter G del buffer
Depositando el cáracter K en el buffer
Recogiendo el carácter K del buffer
Depositando el cáracter H en el buffer
Recogiendo el carácter H del buffer
Programa finalizado
BUILD SUCCESSFUL (total time: 48 seconds)


Projects
Output - Actividad2.1 (run)
run:
Depositando el cáracter B en el buffer
Depositando el cáracter G en el buffer
Recogiendo el carácter G del buffer
Depositando el cáracter L en el buffer
Recogiendo el carácter L del buffer
Depositando el cáracter Z en el buffer
Depositando el cáracter T en el buffer
Recogiendo el carácter T del buffer
Recogiendo el carácter Z del buffer
Recogiendo el carácter B del buffer
Depositando el cáracter J en el buffer
Recogiendo el carácter J del buffer
Depositando el cáracter A en el buffer
Recogiendo el carácter A del buffer
Depositando el cáracter A en el buffer
Recogiendo el carácter A del buffer
Depositando el cáracter Z en el buffer
Recogiendo el carácter Z del buffer
Depositando el cáracter R en el buffer
Recogiendo el carácter R del buffer
Depositando el cáracter I en el buffer
Depositando el cáracter N en el buffer
Recogiendo el carácter N del buffer
Depositando el cáracter T en el buffer
Recogiendo el carácter T del buffer
Recogiendo el carácter I del buffer
Depositando el cáracter P en el buffer
Recogiendo el carácter P del buffer
Depositando el cáracter D en el buffer
Recogiendo el carácter D del buffer
Programa finalizado
BUILD SUCCESSFUL (total time: 51 seconds)


Projects
Output - Actividad2.1 (run)
Main.java
run:
Depositando el cáracter U en el buffer
Depositando el cáracter W en el buffer
Depositando el cáracter G en el buffer
Depositando el cáracter E en el buffer
Recogiendo el carácter E del buffer
Recogiendo el carácter G del buffer
Depositando el cáracter Y en el buffer
Depositando el cáracter K en el buffer
Recogiendo el carácter K del buffer
Recogiendo el carácter Y del buffer
Depositando el cáracter V en el buffer
Recogiendo el carácter V del buffer
Recogiendo el carácter W del buffer
Depositando el cáracter Z en el buffer
Depositando el cáracter R en el buffer
Recogiendo el carácter R del buffer
Recogiendo el carácter Z del buffer
Recogiendo el carácter U del buffer
Depositando el cáracter E en el buffer
Recogiendo el carácter E del buffer
Depositando el cáracter S en el buffer
Recogiendo el carácter S del buffer
Depositando el cáracter X en el buffer
Depositando el cáracter C en el buffer
Recogiendo el carácter C del buffer
Depositando el cáracter A en el buffer
Depositando el cáracter G en el buffer
Recogiendo el carácter G del buffer
Recogiendo el carácter A del buffer
Recogiendo el carácter X del buffer
Programa finalizado
BUILD SUCCESSFUL (total time: 49 seconds)

💛 Ejercicio 2

De igual manera a lo visto en el tema, ahora te proponemos que resuelvas el clásico problema denominado "La cena de los filósofos" utilizando la clase Semaphore del paquete java.util.concurrent.

El problema es el siguiente: Cinco filósofos se sientan alrededor de una mesa y pasan su vida comiendo y pensando. Cada filósofo tiene un plato de arroz chino y un palillo a la izquierda de su plato. Cuando un filósofo quiere comer arroz, cogerá los dos palillos de cada lado del plato y comerá. El problema es el siguiente: establecer un ritual (algoritmo) que permita comer a los filósofos. El algoritmo debe satisfacer la exclusión mutua (dos filósofos no pueden emplear el mismo palillo a la vez), además de evitar el interbloqueo y la inanición.

Para realizar esta actividad deberemos crear tres clases "*Main*", "*Filosofo*" y "*Mesa*". Primero crearemos la clase "*Mesa*", declararemos las variables y crearemos el constructor, este recibirá como parámetro el número de palillos y tendrá un bucle "*for*" que se ejecutará desde 0 hasta "*numPalillos* - 1". Cada vez que se ejecute, se crea un nuevo "*Semaphore*" y se guarda en la posición correspondiente del array "palillos".

```
public class Mesa {
    //DECLARAMOS LAS VARIABLES
    private Semaphore[] palillos;

    //CREAMOS EL MÉTODO Mesa EL CUAL PEDIRÁ COMO ARGUMENTO EL NÚMERO DE PALILLOS
    public Mesa(int numPalillos){
        //CREAMOS UN NUEVO array Semaphore USANDO EL NÚMERO DE PALILLOS COMO LONGITIUD
        this.palillos = new Semaphore[numPalillos];

        //CREAMOS BUCLE for
        for (int i = 0; i < numPalillos; i++) {
            //SE CREA UN NUEVO Semaphore Y SE GUARDA EN LA POSICIÓN CORRESPONDIENTE
            this.palillos[i] = new Semaphore(1);
        }
    }
```

A hora creamos el método "*palilloI*", que devolverá el mismo entero que se le pase como argumento y el método "*palilloD*", que comprueba el entero, si este es igual a 0 retorna el número de palillos que hay menos uno y si no retorna el entero que le pasamos como argumento menos uno.

```
//CREAMOS EL MÉTODO palilloI EL CUAL PEDIRÁ COMO PARÁMETRO UN ENTERO
public int palilloI(int i) {
    //DEVOLBEMOS LA VARIABLE i
    return i;
}

//CREAMOS EL MÉTODO palilloD EL CUAL PEDIRÁ COMO PARÁMETRO UN ENTERO
public int palilloD(int i) {
    //SI LA VARIABLE i ES IGUAL A 0 DEVOLVEMOS EL NÚMERO DE PALILLOS QUE HAY MENOS 1
    if(i == 0){
        return this.palillos.length - 1;
    //SI NO SE CUMPLE LA CONDICIÓN DEVOLVEMOS EL VALOR DE i MENOS 1
    }else{
        return i - 1;
    }
}
```

Ahora creamos el método "*cogerPalillo*", este usa el método "*acquire*" para que el filosofo adquiera primero el palillo izquierdo y luego el derecho.

```
//CREAMOS EL MÉTODO cogerPalillo EL CUAL PEDIÁ EL FILÓSOFO COMO PARÁMETRO
public void cogerPalillo(int filosofo) throws InterruptedException{
    //USAMOS acquire PARA QUE EL FILÓSOFO ADQUIERA EL PALILLO IZQUIERDO
    this.palillos[this.palilloI(filosofo)].acquire();
    //USAMOS acquire PARA QUE EL FILÓSOFO ADQUIERA EL PALILLO DERECHO
    this.palillos[this.palilloD(filosofo)].acquire();
}
```

Y por último creamos el método "dejarPalillo", este usa el método "release" para que el filosofo libere primero el palillo izquierdo y luego el derecho.

```
//CREAMOS EL MÉTODO dejarPalillo EL CUAL PEDIA EL FILÓSOFO COMO PARÁMETRO
public void dejarPalillo(int filosofo){
    //USAMOS release PARA QUE EL FILÓSOFO LIBERE EL PALILLO IZQUIERDO
    this.palillos[this.palilloI(filosofo)].release();
    //USAMOS release PARA QUE EL FILÓSOFO LIBERE EL PALILLO DERECHO
    this.palillos[this.palilloD(filosofo)].release();
}
```

La segunda clase que crearemos será la clase "*Filosofo*"(esta clase extenderá de "Thread"). Lo primero que haremos será declarar las variables y crear el constructor, este recibirá como parámetro un objeto "Mesa" y el número de comensales.

```
public class Filosofo extends Thread{
    //DECLARAMOS LAS VARIABLES
    private Mesa mesa;
    private int filosofo;
    private int pFilosofo;

    //CREAMOS EL CONSTRUCTOR
    public Filosofo(Mesa mesa, int comensal){
        this.filosofo = comensal;
        this.pFilosofo = comensal - 1;
        this.mesa = mesa;
    }
```

Ahora creamos el método "*pensando*", que mostrará por pantalla que filosofo esta pensando y detendrá la ejecución del hilo durante 2 segundos.

```
//CREAMOS EL MÉTODO pensando
public void pensando() throws InterruptedException{
    //MOSTRAMOS POR PANTALLA QUE FILOSOFO ESTA PENSANDO
    System.out.println("Filósofo " + this.filosofo + " Pensando");
    //DETENEMOS LA EJECUCIÓN DEL HILO DURANTE 2 SEGUNDOS
    Thread.sleep(2000);
}
```

Creamos el método "*comiendo*", que mostrará por pantalla que filosofo esta comiendo y detendrá la ejecución del hilo durante 2 segundos.

```
//CREAMOS EL MÉTODO comiendo
public void comiendo() throws InterruptedException{
    //MOSTRAMOS POR PANTALLA QUE FILOSOFO ESTA COMIENDO
    System.out.println("Filosofo " + this.filosofo + " Comiendo");
    //DETENEMOS LA EJECUCIÓN DEL HILO DURANTE 2 SEGUNDOS
    Thread.sleep(2000);
}
```

Y creamos el método "*run*", este tendrá un bucle "*while*" infinito, o de este llamamos al método "*pensando*", mostramos por pantalla que filosofo esta hambriento, llamamos al método "*cogerpalillo*" pasándole como parámetro la posición del filósofo, llamamos al método "*comiendo*", mostramos que filósofo termina de comer y que palillos quedan libres y por último llamamos al método "*dejarpalillo*" pasándole como parámetro la posición del filósofo.

En resumen, este código simula el comportamiento de un filósofo, donde los filósofos deben alternar entre pensar y comer utilizando los palillos.

```
//CREAMOS EL MÉTODO run
public void run(){
    //CREAMOS UN BUCLE while INFINITO
    while(true){
        try {
            //LLAMAMOS AL MÉTODO pensando
            this.pensando();
            //MOSTRAMOS POR PANTALLA QUE FILOSOFO ESTA HAMBRIENTO
            System.out.println("Filósofo " + this.filosofo + " Hambriento");
            //LLAMAMOS AL MÉTODO cogerPalillo PASANDOLE COMO PARÁMETRO LA POSICIÓN DEL FILÓSOFO
            mesa.cogerPalillo(this.pFilosofo);
            //LLAMAMOS AL MÉTODO COMIENDO
            this.comiendo();
            //MOSTRAMOS QUE FILÓSOFO TERMINA DE COMER Y QUE PALILLOS QUEDAN LIBRES
            System.out.println("Filósofo " + this.filosofo + " Termina de comer, Libres palillos: " +
                    (this.mesa.palilloI(this.pFilosofo) + 1) + ", " + (this.mesa.palilloD(this.pFilosofo) + 1));
            //LLAMAMOS AL MÉTODO dejarPalillo PASANDOLE COMO PARÁMETRO LA POSICIÓN DEL FILÓSOFO
            mesa.dejarPalillo(this.pFilosofo);
        //SI SE PRODUCE UNA EXCEPCIÓN EL catch LO CAPTURA Y MUESTRA EL ERROR Y EL stack trace
        } catch (InterruptedException ex) {
            Logger.getLogger(Filosofo.class.getName()).log(Level.SEVERE, null, ex);
        }

    }
}
```

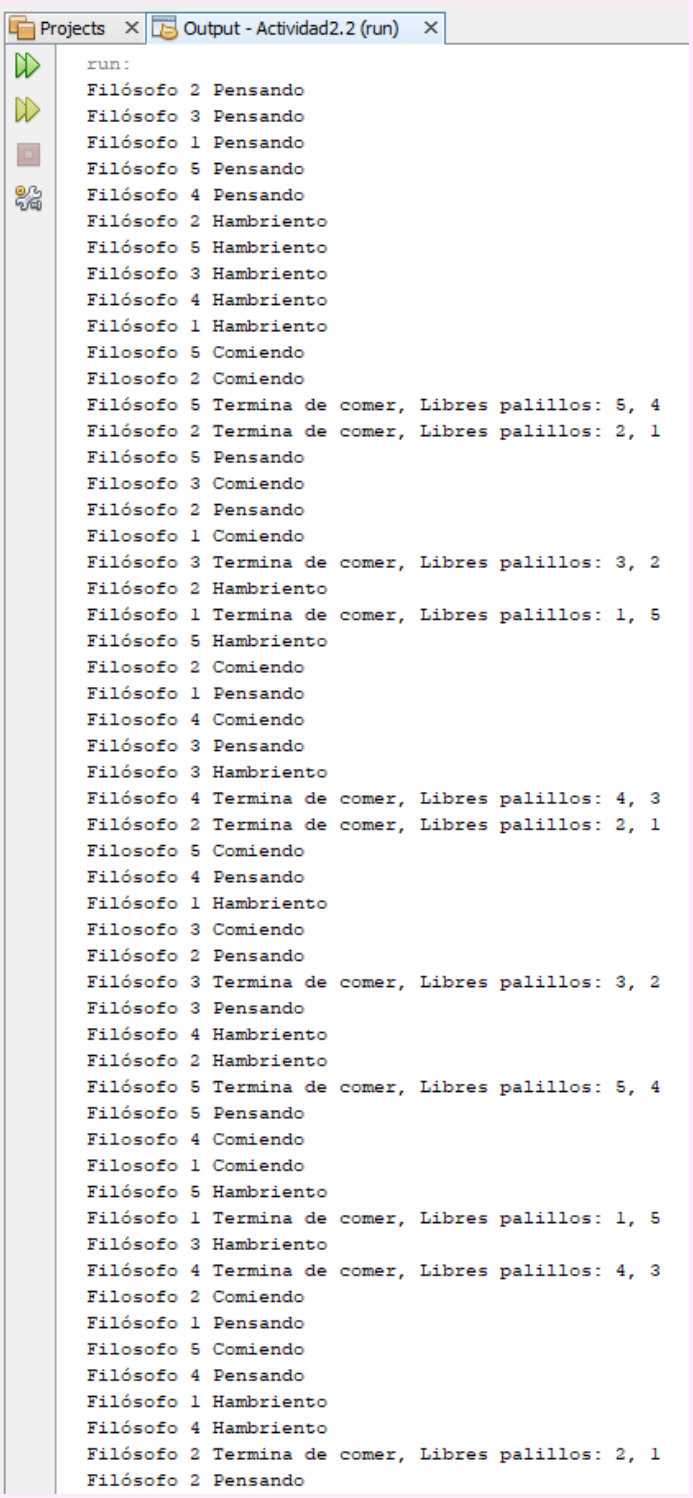
Por último creamos la clase “*Main*”, en ella creamos un entero con el número de filósofos que hay y una instancia de la clase “*Mesa*”, a la cuál le pasaremos como parámetro el número de filósofos.

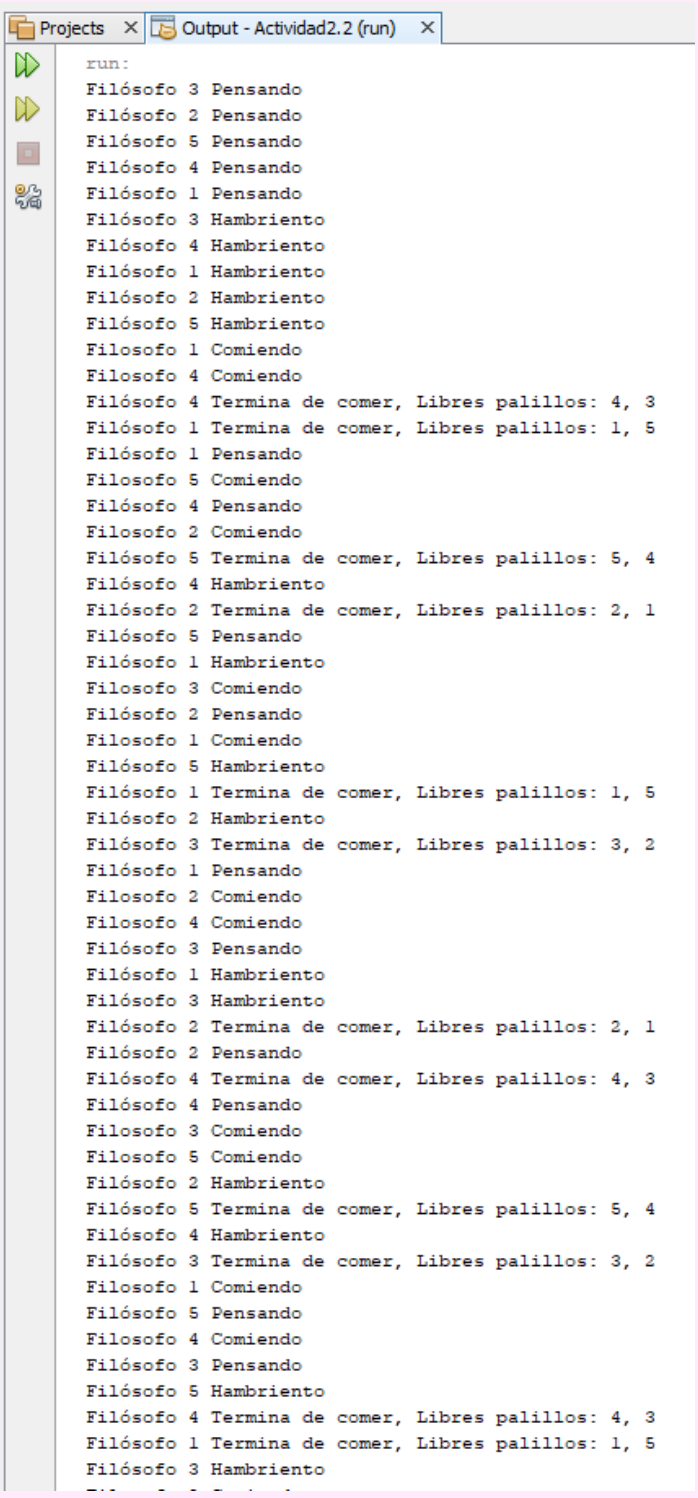
Creamos un bucle “*for*” que ejecutará un número de hilos igual a la cantidad de filósofos, cada vez que se ejecute el bucle creará una instancia de la clase “*filosofo*” pasándole como parámetro la instancia de “Mesa” y el número de la posición de cada filosofo, por último ejecutará el método “*start*” del hilo “*filosofo*” para iniciar su ejecución.

En resumen, este código crea un conjunto de hilos que representan a los filósofos en una mesa, donde cada filósofo intenta coger palillos para comer.

```
public class Main {
    public static void main(String[] args) {
        //CREAMOS UN ENTERO CON EL NÚMERO DE FILOSOFOS
        int numFilosofos = 5;

        //CREAMOS UNA INSTANCIA DE LA CLASE Mesa PASÁNDOLE COMO PARÁMETRO EL NÚMERO DE FILÓSOFOS
        Mesa mesa = new Mesa(numFilosofos);

        //CREAMOS UN BUCLE for QUE EJECUTARÁ UN NÚMERO DE HILOS IGUAL A LA CANTIDAD DE FILOSOFOS
        for (int i = 1 ; i < numFilosofos + 1; i++) {
            //CREAMOS UNA INSTANCIA DE LA CLASE Filosofo PASÁNDOLE COMO PARÁMETRO LA INSTANCIA
            //DE Mesa Y EL NÚMERO DE LA POSICIÓN DE CADA FILOSOFO
            Filosofo filosofo = new Filosofo(mesa, i);
            //EJECUTAMOS EL MÉTODO start DEL HILO  filosofo PARA INICIAR SU EJECUCIÓN
            filosofo.start();
        }
    }
}
```

Pruebas


Projects
Output - Actividad2.2 (run)
run:
Filósofo 2 Pensando
Filósofo 3 Pensando
Filósofo 1 Pensando
Filósofo 5 Pensando
Filósofo 4 Pensando
Filósofo 2 Hambriento
Filósofo 5 Hambriento
Filósofo 3 Hambriento
Filósofo 4 Hambriento
Filósofo 1 Hambriento
Filosofo 5 Comiendo
Filosofo 2 Comiendo
Filósofo 5 Termina de comer, Libres palillos: 5, 4
Filósofo 2 Termina de comer, Libres palillos: 2, 1
Filósofo 5 Pensando
Filosofo 3 Comiendo
Filósofo 2 Pensando
Filosofo 1 Comiendo
Filósofo 3 Termina de comer, Libres palillos: 3, 2
Filósofo 2 Hambriento
Filósofo 1 Termina de comer, Libres palillos: 1, 5
Filósofo 5 Hambriento
Filosofo 2 Comiendo
Filósofo 1 Pensando
Filosofo 4 Comiendo
Filósofo 3 Pensando
Filósofo 3 Hambriento
Filósofo 4 Termina de comer, Libres palillos: 4, 3
Filósofo 2 Termina de comer, Libres palillos: 2, 1
Filosofo 5 Comiendo
Filósofo 4 Pensando
Filósofo 1 Hambriento
Filosofo 3 Comiendo
Filósofo 2 Pensando
Filósofo 3 Termina de comer, Libres palillos: 3, 2
Filósofo 3 Pensando
Filósofo 4 Hambriento
Filósofo 2 Hambriento
Filósofo 5 Termina de comer, Libres palillos: 5, 4
Filósofo 5 Pensando
Filosofo 4 Comiendo
Filosofo 1 Comiendo
Filósofo 5 Hambriento
Filósofo 1 Termina de comer, Libres palillos: 1, 5
Filósofo 3 Hambriento
Filósofo 4 Termina de comer, Libres palillos: 4, 3
Filosofo 2 Comiendo
Filósofo 1 Pensando
Filosofo 5 Comiendo
Filósofo 4 Pensando
Filósofo 1 Hambriento
Filósofo 4 Hambriento
Filósofo 2 Termina de comer, Libres palillos: 2, 1
Filósofo 2 Pensando


Projects
Output - Actividad2.2 (run)
run:
Filósofo 3 Pensando
Filósofo 2 Pensando
Filósofo 5 Pensando
Filósofo 4 Pensando
Filósofo 1 Pensando
Filósofo 3 Hambriento
Filósofo 4 Hambriento
Filósofo 1 Hambriento
Filósofo 2 Hambriento
Filósofo 5 Hambriento
Filosofo 1 Comiendo
Filosofo 4 Comiendo
Filósofo 4 Termina de comer, Libres palillos: 4, 3
Filósofo 1 Termina de comer, Libres palillos: 1, 5
Filósofo 1 Pensando
Filosofo 5 Comiendo
Filósofo 4 Pensando
Filosofo 2 Comiendo
Filósofo 5 Termina de comer, Libres palillos: 5, 4
Filósofo 4 Hambriento
Filósofo 2 Termina de comer, Libres palillos: 2, 1
Filósofo 5 Pensando
Filósofo 1 Hambriento
Filosofo 3 Comiendo
Filósofo 2 Pensando
Filosofo 1 Comiendo
Filósofo 5 Hambriento
Filósofo 1 Termina de comer, Libres palillos: 1, 5
Filósofo 2 Hambriento
Filósofo 3 Termina de comer, Libres palillos: 3, 2
Filósofo 1 Pensando
Filosofo 2 Comiendo
Filosofo 4 Comiendo
Filósofo 3 Pensando
Filósofo 1 Hambriento
Filósofo 3 Hambriento
Filósofo 2 Termina de comer, Libres palillos: 2, 1
Filósofo 2 Pensando
Filósofo 4 Termina de comer, Libres palillos: 4, 3
Filósofo 4 Pensando
Filosofo 3 Comiendo
Filosofo 5 Comiendo
Filósofo 2 Hambriento
Filósofo 5 Termina de comer, Libres palillos: 5, 4
Filósofo 4 Hambriento
Filósofo 3 Termina de comer, Libres palillos: 3, 2
Filosofo 1 Comiendo
Filósofo 5 Pensando
Filosofo 4 Comiendo
Filósofo 3 Pensando
Filósofo 5 Hambriento
Filósofo 4 Termina de comer, Libres palillos: 4, 3
Filósofo 1 Termina de comer, Libres palillos: 1, 5
Filósofo 3 Hambriento

Conclusiones.

En el **ejercicio 1,** mi conclusión es que es posible implementar un sistema de productores-consumidores utilizando hilos en Java. En este caso, se utiliza un búfer compartido con capacidad limitada para almacenar y consumir datos. También se puede observar que se necesita implementar mecanismos de sincronización para garantizar que el productor solo pueda poner datos en el búfer cuando esté vacío o haya espacio, y que el consumidor solo pueda consumir datos cuando el búfer no esté vacío.

En el **ejercicio 2,** mi conclusión es que es posible resolver el problema de "La cena de los filósofos" utilizando la clase Semaphore de Java. En este caso, se utiliza Semaphore para garantizar la exclusión mutua en el acceso a los palillos y evitar interbloqueos y la inanición de los filósofos. Este problema es un ejemplo clásico de cómo utilizar semáforos para resolver situaciones de concurrencia.

Recursos.

Recursos necesarios para realizar la Tarea:

- IDE NetBeans.
- Contenidos de la unidad.
- Ejemplos expuestos en el contenido de la unidad.

Bibliografía.

https://www.biblia.es/biblia-online.php

https://www.churchofjesuschrist.org/study/scriptures?lang=spa

https://pastoralsj.org/biblia

https://www.biblija.net/biblija.cgi?l=es

https://www.bibliatodo.com/la-biblia