Facultad de
Cs. de la Computación

Práctica #1

Benemérita Universidad Autónoma de Puebla

Alumnos:

Luis Angel Avendaño Avalos – 201933893

Christian Amauri Amador Ortega – 201927821

Programación Distribuida

Otoño 2023

Practica 1.

Introducción.

En el contexto de la programación concurrente y distribuida, la creación de sistemas que gestionan recursos compartidos y operaciones simultáneas es un desafío crucial. La presente práctica se centra en el desarrollo de un programa en el lenguaje de programación Java que simula la descarga de objetos de un contenedor mediante dos brazos.

Sustento Teórico

La programación concurrente es esencial en situaciones en las que varias tareas deben realizarse en paralelo, compartiendo recursos y accediendo a datos compartidos. Sin embargo, esta concurrencia también puede dar lugar a problemas como las condiciones de carrera y la incoherencia de datos si no se manejan adecuadamente. Para abordar estos problemas, se requiere una sólida comprensión de los conceptos y las técnicas relacionadas con la sincronización y la exclusión mutua.

Sincronización y Exclusión Mutua: Cuando múltiples hilos acceden a recursos compartidos, es fundamental garantizar que los hilos no interfieran entre sí. El uso de la palabra clave synchronized en Java permite crear bloques de código que solo un hilo puede ejecutar a la vez, lo que garantiza la exclusión mutua y evita condiciones de carrera. Los métodos sincronizados y los bloques de código sincronizados se utilizan para asegurar que los hilos accedan a recursos compartidos de manera ordenada y controlada.

Coordinación y Comunicación: En un entorno donde los hilos trabajan en conjunto, la coordinación y la comunicación son esenciales. Java proporciona métodos como wait(), notify() y notifyAll() para permitir que los hilos esperen y se notifiquen mutuamente sobre eventos importantes. Estos mecanismos de coordinación son cruciales para evitar situaciones en las que un hilo podría estar esperando indefinidamente o para asegurarse de que los hilos se activen cuando sea necesario.

En esta práctica, se implementará un modelo de simulación en el que dos brazos descargan objetos de un contenedor. Se utilizarán hilos para representar los brazos y se aplicarán conceptos de sincronización y coordinación.

Objetivo.

El objetivo de esta práctica es reconocer conceptos clave como la sincronización, la exclusión mutua y la cooperación entre hilos en un entorno distribuido al igual que explorar cómo se puede lograr una ejecución segura y ordenada en un entorno concurrente utilizando Java y las técnicas de sincronización adecuadas. La comprensión de estos conceptos es esencial para desarrollar aplicaciones eficientes y robustas en un mundo cada vez más orientado a la computación distribuida y paralela.

Desarrollo.

Parte I.

Las especificaciones de la presente practica son las siguientes:

El objetivo consiste en implementar un sistema que permita vaciar un contenedor. Este sistema presenta un problema de sección crítica, es importante identificarla y utilizar el algoritmo de Peterson para la solución.

El sistema se compone de dos brazos robots y un contenedor con piezas, compartido por ambos. Cada brazo tiene como propósito tomar una determinada cantidad de piezas del contenedor, de manera que entre los dos brazos robots logren vaciar el contenedor. Los brazos pueden tomar la misma o diferente cantidad de piezas.

En un principio, el contenedor cuenta con 50 piezas y con las siguientes restricciones:

- Un brazo sólo puede tomar una pieza cada vez que accede al contenedor.
- Para evitar colisiones entre los brazos, el acceso al contenedor debe ser en exclusión mutua, es decir, los dos brazos no pueden descargar piezas del contenedor simultáneamente.

Para implementar el sistema se deberán crear tres clases: Contenedor, Brazo y Sistema.

De esta manera, se modelan las 3 clases necesarias quedando de la siguiente manera:

Contenedor.

La clase Contenedor representa el recurso compartido por los dos brazos y es el objeto del proceso de vaciado. Se le presenta un constructor de manera que a este se le pueda ser asignado una capacidad, la cual será el total de piezas con el que el contenedor va a iniciar.

```
private int cantidadObjetos;
public Contenedor(int capacidad) {
    this.cantidadObjetos = capacidad;
}
```

Después se especifica el método de “descarga” en el que se realiza la función precisamente de descargar una pieza del contenedor, esta hace una verificación con un método auxiliar llamado “estaVacio” el cual sirve como verificación para saber si precisamente el contenedor está vacio, si no lo está entonces, si la cantidad de objetos es mayor que cero (o sea un número posible), se decrementa la cantidad de objetos que quedan en el contenedor y se retorna el objeto que fue sacado para que se pueda identificar.

```
public synchronized int descarga(String idBrazo) {
    if (!estaVacio()) { //si el contenedor no está vacio
        if (cantidadObjetos > 0) {//mientras no sea cero
            cantidadObjetos--;
            return cantidadObjetos + 1;//devuelve num de objetos que quedan en el contenedor
        }
    }
```

De otra manera, si el contenedor está vacío lo que hace el programa es que manda un print con el aviso y utiliza un wait en el contenedor, desbloqueando así el recurso compartido para otros hilos desbloqueándolos y bloqueando el hilo actual.

```
        System.out.println("Brazo" + idBrazo + ": Contenedor vacio, espere...");
        try {
            wait();
        } catch (InterruptedException ex) {
            Logger.getLogger(Contenedor.class.getName()).log(Level.SEVERE, null, ex);
        }
        return -1; // No debería ocurrir si se verifica si está vacío antes de sacar
    }
```

Brazo.

La clase Brazo implementa una abstracción del brazo robot. Cada brazo es un proceso en el sistema concurrente, por lo que, esta clase deberá implementar la interfaz Runnable. El constructor de la clase tendrá tres parámetros de entrada: un identificador único asignado al brazo, el número de piezas que deben ser tomadas por el brazo durante su actividad y, finalmente, el contenedor sobre el que debe trabajar. Se toma un atributo extra el cual es un contador para saber cuantas piezas a sacado en su proceso el cual se inicializa en cero.

```
class Brazo extends Thread {
    private String idBrazo;
    private Contenedor contenedor; //recurso compartido
    private int piezasExtrae;//piezas que puede extraer en su totalidad
    private int contPiezas;//contador de cuantas piezas ha sacado

    public Brazo(String nombre, int piezas, Contenedor contenedor) {
        this.idBrazo = nombre;
        this.contenedor = contenedor;
        this.piezasExtrae=piezas;
        this.contPiezas=0;
    }
```

En su interfaz Run se ejecuta el procedimiento de sacar una pieza por parte del brazo, de manera que de primeras se sincroniza el recurso compartido que es el contenedor bloqueándolo para que solo el hilo actual pueda utilizarlo, por siguiente lo primero que hace es verificar (con ayuda de la función “verifica”) si el brazo ha llegado a su limite de piezas que puede sacar durante el proceso entonces se notifica con un print dejando saber esto y cuántas piezas aun quedan en el contenedor y por consiguiente deja esperando al hilo.

Por el contrario, si el brazo no ha llegado a su límite entonces se realiza el proceso de descarga sobre el contenedor, se incremente el contador de piezas sacadas y se printea qué brazo saca qué pieza y cuántas piezas ha sacado en su totalidad ese brazo. Finalmente, con notifyAll se desbloquea el recurso compartido a otros hilos y se bloquea este hilo.

```java
public void run() {
    while (true) {
        synchronized (contenedor) {
            if(!verifica()){
                try {
                    System.out.print("Brazo "+idBrazo+": Alcanzo su limite, sacó "+contPiezas+" piezas");
                    System.out.println(", quedan "+contenedor.getcantidadObjetos()+" piezas en el contenedor");
                    contenedor.wait();
                } catch (InterruptedException ex) {
                    Logger.getLogger(name: Brazo.class.getName()).log(level: Level.SEVERE, msg:null, thrown: ex);
                }
            }else{
                int objeto = contenedor.descarga(idBrazo);
                contPiezas++;
                System.out.println("Brazo "+idBrazo + ": Sacó la pieza "+objeto+"\nHa sacado "+contPiezas+" piezas");
                contenedor.notifyAll();
            }
        }
        try {
            Thread.sleep(millis: 100); // Simulación de tiempo de descarga
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

Sistema.

Finalmente, la clase Sistema es la aplicación que configura y ejecuta el sistema. Contendrá el método main() donde serán declarados los elementos que constituyen el sistema: un contenedor de 50 piezas y dos brazos robots que tendrán como propósito vaciarlo. Inicialmente, cada brazo puede estar configurado para tomar 25 piezas del contenedor, aunque este parámetro puede ser posteriormente modificado.

```java
public class Prct1 {
    public static void main(String[] args) {
        Contenedor contenedor = new Contenedor(capacidad: 50);//recurso compartido

        Brazo brazo1 = new Brazo(nombre: "A", piezas: 30,contenedor);
        Brazo brazo2 = new Brazo(nombre: "B", piezas: 30, contenedor);


        brazo1.start();
        brazo2.start();
    }
```

Parte II.

Las especificaciones de la segunda parte de la practica son las siguientes:

implementar un sencillo sistema de producción compuesto por dos contenedores compartidos y dos brazos robots capaces de tomar piezas de los contenedores de manera coordinada para producir unidades de un producto.

Detrás de cada brazo hay un operario que tiene como objetivo montar unidades de un producto. Para producir cada unidad se necesita una pieza de cada contenedor. Así que, el Brazo 1 primero trata de tomar una pieza del Contenedor A y después del Contenedor B. Una vez que dispone de ambas piezas el operario ensambla el producto final. El funcionamiento del Brazo 2 es similar, pero el orden en el acceso a los contenedores es diferente. En este caso el brazo primero toma una pieza del Contenedor B y después del Contenedor A.

Restricciones adicionales:

- Un brazo solo es capaz de tomar una pieza cada vez que accede al contenedor.
- Para evitar colisiones entre los brazos, el acceso a cada contenedor es en exclusión mutua, es decir, los dos brazos no pueden simultáneamente descargar piezas de un mismo contenedor.

Implementamos entonces el sistema con las tres clases actualizadas: Contenedor, BrazoProduccion y SistemaProduccion.

Contenedor.

Representa un contenedor que almacena las piezas necesarias para la producción de los productos finales. La clase mantiene un registro de la cantidad de piezas disponibles en el contenedor y proporciona métodos para un acceso seguro a estas piezas.

```
class Contenedor {
    private int cantidadObjetos;
    private String ID;

    public Contenedor(int capacidad,String ID) {
        this.cantidadObjetos = capacidad;
        this.ID = ID;
    }

    public int getCantidadObjetos() {
        return cantidadObjetos;
    }

    public String getID() {
        return ID;
    }
```

El método descarga() permite que los brazos tomen piezas del contenedor, utilizando sincronización para evitar conflictos cuando varios brazos intentan acceder simultáneamente. Si el contenedor está vacío, este método pone al brazo en espera hasta que haya piezas disponibles nuevamente.

```
public synchronized int descarga(String idBrazo, String tipo) {
    if (!estaVacio()) {
        cantidadObjetos--;
        return cantidadObjetos + 1;
    }

    System.out.println("Brazo " + idBrazo + ": Contenedor " + tipo + " vacio, esperando..

    try {
        wait();
    } catch (InterruptedException ex) {
        Logger.getLogger(Contenedor.class.getName()).log(Level.SEVERE, null, ex);
    }

    return -1;
}
```

La clase también incluye el método estaVacio() que verifica si el contenedor está sin piezas.

```
public synchronized boolean estaVacio() {
    return cantidadObjetos == 0;
}
```

BrazoProduccion.

Representa un brazo de producción en el sistema. Cada brazo tiene su identificador único (idBrazo) y una cantidad máxima de piezas que puede ensamblar (piezasExtrae), además de los dos contenedores sobre los que trabajará (contenedor1, contenedor2).

```
public class BrazoProduccion extends Thread {
    private String idBrazo;
    private Contenedor contenedor1;
    private Contenedor contenedor2;
    private int piezasExtrae;
    private int contPiezas;

    public BrazoProduccion(String nombre, int piezas, Contenedor contenedor1, Contenedor con
        this.idBrazo = nombre;
        this.contenedor1 = contenedor1;
        this.contenedor2 = contenedor2;
        this.piezasExtrae = piezas;
        this.contPiezas = 0;
    }

    public boolean verifica() {
        return contPiezas < piezasExtrae;
    }
```

El método run() implementa la lógica de ensamblaje de productos a partir de las piezas de los contenedores. La sincronización se utiliza en este método para garantizar que los brazos accedan a los contenedores de manera segura y eviten conflictos cuando intentan tomar piezas.

```
@Override
public void run() {
    while (verifica()) {
        int objeto1, objeto2;

        synchronized (contenedor1) {
            if (idBrazo.equals("2")) {
                objeto1 = contenedor2.descarga(idBrazo, contenedor2.getID());
            } else {
                objeto1 = contenedor1.descarga(idBrazo, contenedor1.getID());
            }
        }

        synchronized (contenedor2) {
            if (idBrazo.equals("2")) {
                objeto2 = contenedor1.descarga(idBrazo, contenedor1.getID());
            } else {
                objeto2 = contenedor2.descarga(idBrazo, contenedor2.getID());
            }
        }
```

Los brazos realizan una verificación para determinar si pueden ensamblar más productos antes de continuar. El contador contPiezas lleva un registro de las piezas ensambladas por el brazo.

```
        contPiezas++;
        System.out.println("Brazo " + idBrazo + ": Ha tomado la pieza "+objeto1+" de
        System.out.println("Brazo " + idBrazo + ": Ha tomado la pieza "+objeto2+" de
        System.out.println("Brazo " + idBrazo + ": Ha montado su producto " + contPi

        contenedor1.notificar();
        contenedor2.notificar();
    }

    System.out.println("Brazo " + idBrazo + ": Ha alcanzado su limite, ha ensamblado
    System.out.println("Quedan "+contenedor1.getCantidadObjetos()+" piezas en el con

}
```

SistemaProduccion:

Desencadena y supervisa todo el sistema de producción concurrente. En su método main(), esta clase crea instancias de contenedores y brazos, y las inicia para que trabajen juntos en la producción. Es responsable de orquestar la interacción entre los brazos y los contenedores, asegurando que el sistema funcione de manera coordinada y eficiente. Esta clase principal establece el contexto y los parámetros iniciales del sistema de producción antes de su ejecución.

```
public class SistemaProduccion {
    public static void main(String[] args) {
        Contenedor contenedorA = new Contenedor(35,"A");
        Contenedor contenedorB = new Contenedor(35,"B");

        BrazoProduccion brazo1 = new BrazoProduccion("1", 25,contenedorA, contenedorB);
        BrazoProduccion brazo2 = new BrazoProduccion("2", 25,contenedorB, contenedorA);

        brazo1.start();
        brazo2.start();
    }
}
```

Resultados.

Parte I.

Para los efectos de prueba se realizarán 3 casos los cuales son los siguientes:

- El contenedor tiene 50 piezas y cada brazo tiene como limitante el mismo número de piezas, pero la cantidad exacta como para vaciar al contenedor en su totalidad, siendo el limitante de 25 piezas para cada brazo.

```
Contenedor contenedor = new Contenedor(capacidad: 50);//recurso compartido

Brazo brazo1 = new Brazo(nombre: "A", piezas: 25,contenedor);
Brazo brazo2 = new Brazo(nombre: "B", piezas: 25, contenedor);
```

La ejecución es tal que normalmente los brazos van sacando piezas del contenedor, la parte final es la que nos interesa. En los prints se notifica qué pieza saca qué brazo y el contador de cuantas piezas ha sacado dicho brazo. Finalmente, se nos notifica cuando el brazo haya alcanzado su limite y muestra cuantas piezas aun quedan en el contenedor.

```
Brazo A: Sacó la pieza 2
Ha sacado 25 piezas
Brazo B: Sacó la pieza 1
Ha sacado 25 piezas
Brazo B: Alcanzo su limite, sacó 25 piezas, quedan 0 piezas en el contenedor
Brazo A: Alcanzo su limite, sacó 25 piezas, quedan 0 piezas en el contenedor
```

- El siguiente caso es cuando hay más piezas en el contenedor que la suma de los limitantes de ambos brazos.

```
Contenedor contenedor = new Contenedor(capacidad: 60);//recurso compartido

Brazo brazo1 = new Brazo(nombre: "A", piezas: 25,contenedor);
Brazo brazo2 = new Brazo(nombre: "B", piezas: 25, contenedor);
```

  Se observa que llegado al limite de los brazos estos se quedan en espera aún quedando piezas en el contenedor, por lo tanto, es el resultado esperado.

```
Brazo A: Sacó la pieza 12
Ha sacado 25 piezas
Brazo B: Sacó la pieza 11
Ha sacado 25 piezas
Brazo A: Alcanzo su limite, sacó 25 piezas, quedan 10 piezas en el contenedor
Brazo B: Alcanzo su limite, sacó 25 piezas, quedan 10 piezas en el contenedor
```

- El siguiente caso es aquel en el que la suma de los limitantes de ambos brazos es mayor a la cantidad de piezas existentes en el contenedor.

```
Contenedor contenedor = new Contenedor(capacidad: 40);//recurso compartido

Brazo brazo1 = new Brazo(nombre: "A", piezas: 25,contenedor);
Brazo brazo2 = new Brazo(nombre: "B", piezas: 25, contenedor);
```

  En la ejecución se observa que llegado a su debido punto el contenedor quedará vacío pero los brazos no llegarán a su limite aun así estos quedarán en espera pues no hay más piezas a sacar del contenedor.

```
Brazo A: Sacó la pieza 2
Ha sacado 20 piezas
Brazo B: Sacó la pieza 1
Ha sacado 20 piezas
BrazoA: Contenedor vacio, espere...
BrazoB: Contenedor vacio, espere...
```

Parte II.

Para los efectos de prueba se realizarán 3 casos los cuales son los siguientes:

- Cada contenedor tiene 50 piezas y cada brazo tiene como limitante el mismo número de piezas, pero la cantidad exacta como para vaciar al contenedor en su totalidad, siendo el limitante de 25 piezas, de cada contenedor, para cada brazo. (Capacidad de los contenedores igual al límite de piezas de los brazos)

```
Contenedor contenedorA = new Contenedor(50,"A");
Contenedor contenedorB = new Contenedor(50,"B");

BrazoProduccion brazo1 = new BrazoProduccion("1", 25,contenedorA, contenedorB);
BrazoProduccion brazo2 = new BrazoProduccion("2", 25,contenedorB, contenedorA);
```

Cada brazo puede ensamblar su máximo número de productos utilizando todas las piezas disponibles en los contenedores antes de alcanzar su límite. En este escenario, ambos brazos alcanzarán su límite al mismo tiempo.

```
Brazo 1: Ha tomado la pieza 2 del contenedor B
Brazo 1: Ha montado su producto 25 de 25

Brazo 1: Ha alcanzado su limite, ha ensamblado 25 productos.
Quedan 1 piezas en el contenedor 'A', y 1 en el contenedor 'B'.
Brazo 2: Ha montado su producto 24 de 25

Brazo 2: Ha tomado la pieza 1 del contenedor B
Brazo 2: Ha tomado la pieza 1 del contenedor A
Brazo 2: Ha montado su producto 25 de 25

Brazo 2: Ha alcanzado su limite, ha ensamblado 25 productos.
Quedan 0 piezas en el contenedor 'B', y 0 en el contenedor 'A'.
```

- El siguiente caso es cuando hay más piezas en los contenedores que la suma de los limitantes de ambos brazos (55 piezas en “A” + 60 piezas en “B” = 115 piezas en total) (limite de cada brazo = 25 piezas).

```
Contenedor contenedorA = new Contenedor(55,"A");
Contenedor contenedorB = new Contenedor(60,"B");

BrazoProduccion brazo1 = new BrazoProduccion("1", 25,contenedorA, contenedorB);
BrazoProduccion brazo2 = new BrazoProduccion("2", 25,contenedorB, contenedorA);
```

En este caso habrá un excedente de piezas en los contenedores después de que ambos brazos alcancen su límite. Esto significa que los contenedores aún tendrán piezas disponibles una vez que los brazos hayan terminado su trabajo.

```
Brazo 2: Ha tomado la pieza 12 del contenedor A
Brazo 2: Ha montado su producto 24 de 25

Brazo 2: Ha tomado la pieza 6 del contenedor B
Brazo 2: Ha tomado la pieza 11 del contenedor A
Brazo 2: Ha montado su producto 25 de 25

Brazo 2: Ha alcanzado su limite, ha ensamblado 25 productos.
Quedan 10 piezas en el contenedor 'B', y 5 en el contenedor 'A'.
```

- El siguiente caso es aquel en el que la suma de los limitantes de ambos brazos es mayor a la cantidad de piezas totales en ambos contenedores (35 piezas en “A” + 35 piezas en “B” = 70 piezas en total). (limite de cada brazo = 25).

```
Contenedor contenedorA = new Contenedor(35,"A");
Contenedor contenedorB = new Contenedor(35,"B");

BrazoProduccion brazo1 = new BrazoProduccion("1", 25,contenedorA, contenedorB);
BrazoProduccion brazo2 = new BrazoProduccion("2", 25,contenedorB, contenedorA);
```

En este caso, uno de los brazos, o los dos brazos estarán esperando que se repongan las piezas en los contenedores (dependiendo del cuál termine primero, si es que al menos uno puede terminar).

```
Brazo 2: Ha tomado la pieza 1 del contenedor B
Brazo 2: Ha tomado la pieza 1 del contenedor A
Brazo 2: Ha montado su producto 10 de 25

Brazo 2: Contenedor A vacio, esperando...
```

Conclusión.

En este proyecto, se implementó un sistema de producción concurrente con dos brazos de ensamblaje y dos contenedores de piezas. Se aplicó la exclusión mutua para garantizar un acceso seguro a los recursos compartidos. El sistema demostró coordinación efectiva entre los brazos, evitando condiciones de carrera y asegurando la integridad de los recursos compartidos. Este proyecto ilustra la aplicación práctica de conceptos clave de programación concurrente en un entorno de producción.