Universidad de los Andes
Departamento de Ingeniería de Sistemas y Computación

Caso 1

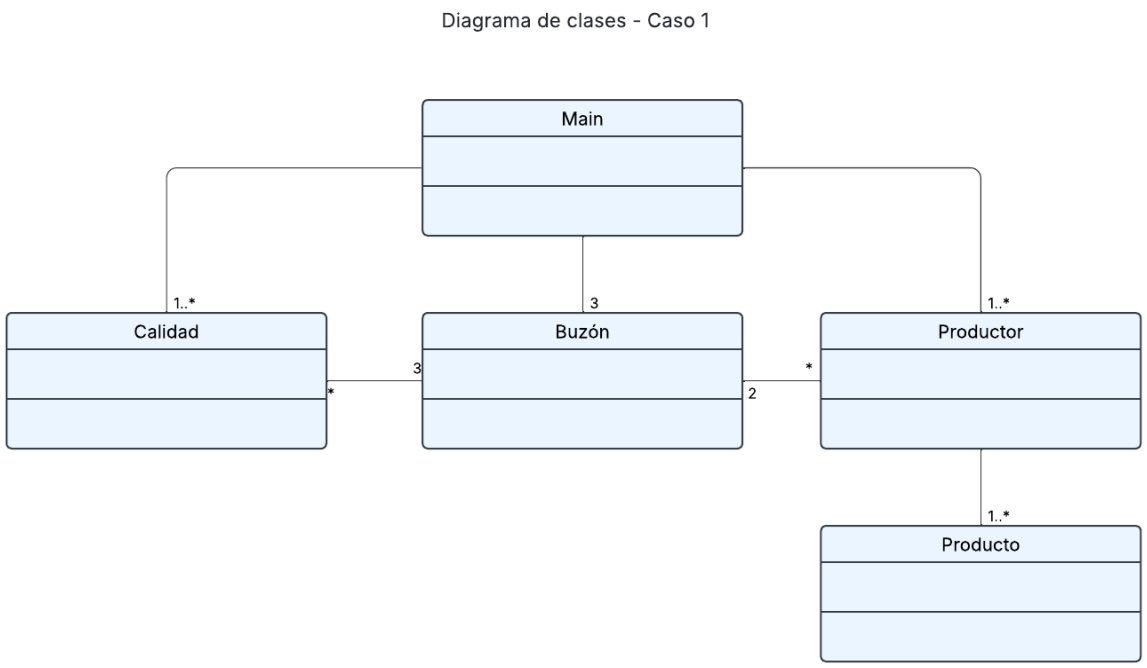
Grupo LET
Julian Mondragon - 202221122
Alejandro Hoyos - 202215277

Infraestructura Computacional
ISIS-2203
Docente Ricardo Gomez Diaz

Bogotá, Colombia

2025-10

Diagrama de clases preliminar



Estrategia para los procesos

Para implementar el caso se desarrollan ciertas estrategias para manejar la concurrencia en un proyecto de simulación de una línea de producción de algunos productos donde se presentan diferentes interacciones con recursos compartidos. Algunas de estas estrategias o mecanismos son:

- Espera semiactiva con el yield()
- Espera activa que va junto con while para verificar constantemente su condición
- Espera pasiva mediante wait() y notify
- Señalamiento mediante la función wait() y notify()
- Algunos métodos de barrera como es el caso del join()
- Monitores para garantizar que se manejen correctamente los recursos compartidos que en este caso sería el buzón
- Métodos con “synchronized” para garantizar que múltiples Threads accedan a la misma sección crítica de manera simultánea

Estructura del programa

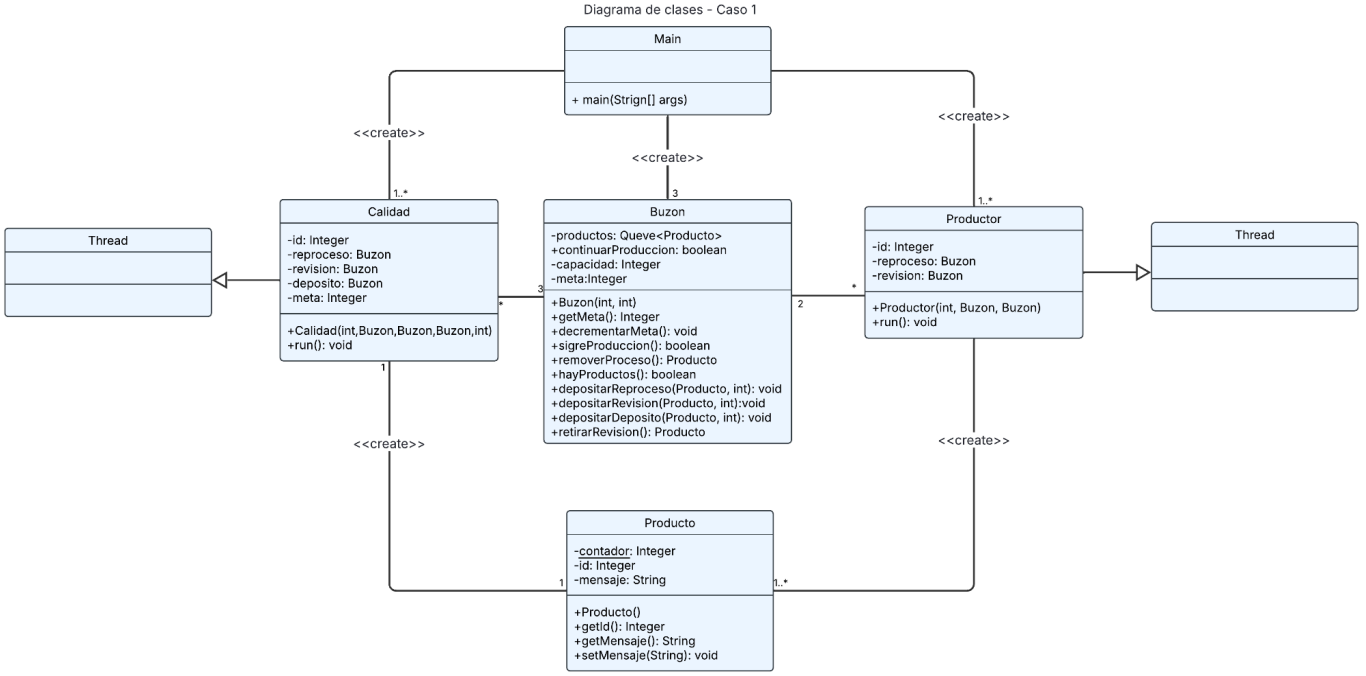
En este caso para la implementación del programa se van a crear 4 clases adicionales a la clase Main, estas 4 clases son:

- **Producto**: Clase que representa el objeto el cual va a ser manipulado por los productores y acto seguido por los buzones y el equipo de calidad. Esta clase no tiene ninguna implementación para ser considerado un Thread
- **Productor**: encargado de generar la productos y de priorizar los productos que se encuentren en el buzón de reprocesos en tal caso de no que el buzón

de reprocesos esté vacío; el encargado de generar los productos será el productor. En este caso los productores son Threads que accederán a los recursos compartidos.

- **Calidad**: es la clase encargada de simular al equipo de calidad pues estos deberán hacer la revisión para aprobar o denegar un producto para pasarlo del buzón de revisión al buzón de depósito. Los del equipo de calidad también son Threads que accederán a los recursos compartidos generando concurrencia.
- **Buzon**: Es la clase encargada de ser el monitor para la gestión de aquellos recursos compartidos, en este caso se tendrían 3 tipos de buzón: buzón de reprocesos, buzón de revisión y buzón de depósito o depósito. Esta clase contendrá los métodos con synchronized para así manejar correctamente el acceso a los recursos compartidos.

Diagrama de clase final



Tras una análisis detallado y después de cambios en enunciado, observamos que el equipo de calidad es capaz de crear un producto que tenga el mensaje “FIN” por lo que es importante hacer este cambio en la relación y en el Diagrama de clases UML

Análisis de Clases

Clase Main

Encargada de pedir datos por consola como el número de operarios, la capacidad que tendrá el buzón de revisión y la meta de los productos que se deben producir. Adicionalmente, en esta clase se inician los monitores que en este caso serían las 3 instancias de Buzón: reproceso, revisión y depósito. Es importante aclarar que el único que tiene un límite o capacidad es el buzón de revisión. Finalmente, se crean e inicializan los operarios (Producto y Calidad) y se esperan hasta que todos terminen para saber cuando el proceso ha sido terminado.

Clase Productor

En la clase producto solo se tiene acceso a los buzones de revisión y de reproceso que serán directamente proporcionados por la clase Main. Esta clase cuenta con el método run que es donde inicia a ejecutar el Thread, en la función run() se encuentra:

- Espera activa: para verificar si se sigue en revisión mediante del buzón de revisión
- Se remueve el producto del buzón de reprocesos e imprime dicha información del producto
- Evalúa si el producto contiene el mensaje “FIN” para detener la producción en el buzón de revisión
- Revisa si el buzón de reprocesos se ha quedado sin productos para que esta clase cree los productos
- Simula el Thread.sleep para depositar el Producto en el buzón de revisión

Clase Calidad

En esta clase de calidad se tiene acceso a los 3 buzones creados directamente en el main() pues se tiene acceso a el buzón de reprocesos para crear el producto con el mensaje de “FIN” y obtener prioridad sobre este buzón. Se tiene acceso al buzón de revisión para aprobar o denegar el paso de los productos al depósito. Cómo esta clase también es una extensión del Thread cuenta con un método run() en donde se incluye:

- Variable interna para revisar la verificar la cantidad de fallos totales que corresponde al 10% de la meta
- Espera activa en donde se verifica que se siga operando en el buzón de revisión o si se encuentran productos y acto seguido verificar y consultar en el buzón de revisión de manera semiactiva con el yield()
- Se retira el producto relacionado y se revisa si falla o no: esto se hace con la función random y si el módulo de 7 es igual a 0 significa que era un múltiplo de 7 por lo que ese producto fallaría en sí y se devolvería al buzón de reprocesos

- En el caso de que haya fallado se disminuye la cantidad de fallos, si la cantidad de fallos es menor o igual a 0 significa que todos los productos que sigan deben ser aprobados
- En tal caso de cumplirse la meta se crea el producto con el mensaje FIN para depositar al buzón de reprocesos
- Se simula el tiempo de procesamiento con el Thread.sleep()

Clase Buzón

Es la clase que actúa como monitor para manejar los recursos compartidos por aquel motivo cuenta con diferentes métodos synchronized para controlar que los Threads accedan simultáneamente a los métodos y modifiquen o lean dichos recursos.

- Se cuenta con Queve creada del paquete de java.util.Queue y asociada con LinkedList
- Se cuentan con distintas variables para manejar el estado en el que se encuentra el buzón ya sea que se deba detener la producción o se debe seguir revisando los productos en el buzón de revisión. Para este caso son de tipos booleanos
- Adicionalmente se cuentan con algunos métodos para controlar el acceso a la variable meta que es brindada por el usuario para que los Threads no la modifique y sea consistente en su acceso y modificación
- Hay métodos dirigidos al comportamiento del buzón de reprocesos como es el caso de hayProductos() pues se encarga de revisar si aún quedan productos. Otra función alusiva a el buzón de reprocesos es el método para depositar productos en el buzón de reprocesos un ejemplo de esto es la creación del Producto “FIN”.
- Métodos dirigidos para buzón de revisión como el de retirar y depositar un producto. Para depositar, si se sigue en producción o se excede la capacidad de este buzón se hará un wait() hasta que se libere mediante un notify a medida que se vayan removiendo productos sobre el buzón de revisión. Al depositarlo se notificará a todos los Threads
- Finalmente se tiene solo un método alusivo directamente al comportamiento del Depósito que será determinado por el operario de calidad

Clase Producto

Es el objeto con el cuál se manipulará por cada una de las otras clases e instancias tiene un contador para reconocer cuál será su id que se determinará de forma creciente tal que si se empieza en 0 el primer producto tendrá de id 0+1. Adicionalmente, se tiene un mensaje que caracteriza cada producto.

NOTA: En este caso se tiene en cuenta una lógica para que el monitor sea manejado de forma consistente tal que no se generen inconvenientes con cada operación. Por este motivo se tienen diferentes métodos en el monitor

alusivos a cada tipo de buzón que serán usados solamente por la instancia adecuada.

Descripción de la ejecución

```
//Pedimos el numero de operarios
System.out.println(x:"Introduce el numero de operarios: ");
int numOperarios = sc.nextInt();

//Pedimos la capacidad del buzon de revision
System.out.println(x:"Introduce la capacidad del buzon de revision: ");
int capacidadRevision = sc.nextInt();

//Pedimos la meta de productos a producir
System.out.println(x:"Introduce la meta de productos a producir: ");
int meta = sc.nextInt();

//Creamos los buffer (buzones)
Buzon reproceso = new Buzon(Integer.MAX_VALUE, meta);
Buzon revision = new Buzon(capacidadRevision, meta);
Buzon deposito = new Buzon(Integer.MAX_VALUE, meta);

//Lista de operarios
Productor[] productores = new Productor[numOperarios];
Calidad[] calidad = new Calidad[numOperarios];
```

Figura 1: Inicio de la ejecución

La ejecución del programa comienza en la clase Main. En esta clase, el usuario ingresa el número de operarios necesarios, asegurando que haya la misma cantidad de operarios de producción y de calidad. Además, se crean los tres buzones principales, que funcionan como buffers en el sistema productor-consumidor, y se le asigna al buzón de revisión la capacidad especificada por el usuario.

Asimismo, la meta de productos se almacena dentro de la clase de operarios de calidad, de modo que cada instancia creada conozca la cantidad de productos que deben ser depositados en el almacén. Una vez definidos estos datos, se ejecutan los procesos de cada operario para iniciar el sistema.

```java
@Override
public void run() {
    while (revision.sigueProduccion()) {  // Verificar si la producción sigue activa

        Producto producto = null;

        synchronized (reproceso) {
            if (!revision.sigueProduccion()) break;  // Verificar nuevamente antes de operar

            producto = reproceso.removerReproceso();

            if (producto != null) {
                System.out.println("Productor " + id +
                        " reprocesó producto " + producto.getId());

                // Si recibe FIN, detener producción global
                if ("FIN".equals(producto.getMensaje())) {
                    System.out.println("Productor " + id +
                            " recibió FIN. Deteniendo producción.");
                    revision.detenerProduccion();
                    break;
                }
            } else {

                if (!revision.sigueProduccion()) break;

                producto = new Producto();
                System.out.println("Productor " + id +
                        " creó producto " + producto.getId());
            }
        }
```

Figura 2: Primer bloque sincronizado en la clase Productor

```java
//Metodo para remover un producto del buzon de reproceso
public synchronized Producto removerReproceso() {
    if (productos.isEmpty()) {
        return null;
    }
    Producto producto = productos.poll();
    return producto;
}
```

Figura 3: Método para tomar un producto del buzón de reproceso

El funcionamiento del productor, como se muestra en la Figura 2, depende de un método clave que permite acceder a una variable estática dentro de la clase Buzón. Esta variable es un booleano que indica si la producción debe continuar o detenerse.

El primer bloque sincronizado del Thread productor se encarga de sincronizar el acceso al buzón de reproceso. Esto garantiza la exclusión mutua entre los productores, evitando que múltiples hilos modifiquen la cola del buzón

simultáneamente y asegurando que solo un productor pueda retirar productos a la vez.

Dentro de este bloque, se verifica si algún producto contiene el mensaje FIN, lo que indica que la producción debe detenerse. Si no se encuentra este mensaje, la producción continúa y el producto obtenido del buzón de reproceso se deposita en el buzón de revisión. Este proceso se repite mientras haya productos en el buzón de reproceso.

Como se muestra en la Figura 3, el método correspondiente en la clase Buzón utiliza una condición if para determinar si aún quedan productos en el buzón.

```
try {
    revision.depositarRevision(producto, id);
    Thread.sleep(millis:100);
} catch (InterruptedException e) {
    Thread.currentThread().interrupt();
    break;
}
```

Figura 4: Agregar producto al buzón de revisión

```
//Metodo para depositar un producto en el buzon de revision
public synchronized void depositarRevision(Producto producto, int id) throws InterruptedException {
    while (productos.size() >= capacidad && sigueProduccion()) {
        System.out.println("Productor " + id + " espera espacio en el buzon de revision...");
        wait(); //Espera pasiva sin consumir CPU
    }

    if (sigueProduccion() && productos.size() < capacidad) {
        productos.add(producto);
        System.out.println("Productor " + id +
            " deposita producto " + producto.getId() + " en el buzon de revision");
    }
    notifyAll();
}
```

Figura 5: Método para guardar producto en revisión

Cuando el Buzón ya no contiene productos, como se muestra en la Figura 4, se crea un nuevo producto y se le asigna un identificador automáticamente mediante un contador en la clase Producto. Este nuevo producto es enviado al buzón de revisión.

Como se observa en la Figura 5, este proceso opera mediante una espera pasiva. Si el buzón de revisión está lleno, el productor entra en estado de suspensión hasta que haya espacio disponible. Cuando un hilo libera espacio en el buzón, notifica a los demás productores para que puedan verificar nuevamente la disponibilidad y, en caso de haber espacio, depositar el producto.

```java
public void run() {

    int fallos = (int) Math.floor(meta * 0.1); //Calcula el 10% de la meta de productos

    while (revision.sigueProduccion() || revision.hayProductos()) { //Verifica que la produccion siga activa o que haya productos en el buzon de revision

        while (!revision.hayProductos() && revision.sigueProduccion()) {
            Thread.yield(); //Espera semi-activa
        }

        Producto producto = null;

        synchronized(revision) {
            producto = revision.retirarRevision();
            if (producto == null) {
                continue;
            } else {
                System.out.println("Calidad " + id + " revisa producto " + producto.getId());
            }
        }
```

Figura 6: Primer bloque sincronizado en la clase Calidad

Un segundo aspecto clave en el sistema es el rol de los operarios de calidad. Como se muestra en la Figura 6, su ejecución comienza mediante una espera semi-activa, en la que verifican constantemente la presencia de productos en el buzón de revisión. Si hay productos disponibles, los extraen para su revisión; de lo contrario, permanecen en un bucle infinito.

Una vez que extraen un producto, ingresan a un bloque sincronizado en torno al buzón de revisión. Esto garantiza que, mientras un hilo está retirando productos del buzón, ningún otro pueda agregar o extraer productos simultáneamente, asegurando así la exclusión mutua.

```java
//Revisar si hay fallo o no
synchronized (deposito) {
    if (revision.getMeta() > 0) {
        if ((random.nextInt(bound:100) + 1) % 7 == 0 && fallos > 0) {
            System.out.println("Calidad " + id + " detectó falla en producto " + producto.getId());
            reproceso.depositarReproceso(producto, id);
            fallos--;
        } else {
            System.out.println("Calidad " + id + " aprobó producto " + producto.getId());
            revision.decrementarMeta();
            deposito.depositarDeposito(producto, id);
        }
    } else {
        Producto fin = new Producto();
        fin.setMensaje(mensaje:"FIN");
        reproceso.depositarReproceso(fin, id);
    }
}
```

Figura 7: Segundo bloque sincronizado en la clase Calidad

Como se muestra en la Figura 7, en la segunda parte de su ejecución, los operarios de calidad se sincronizan en torno al depósito. Esto garantiza la exclusión mutua, evitando que múltiples hilos depositen productos simultáneamente.

Dentro de este bloque sincronizado, se genera un número aleatorio para determinar si el producto pasa la inspección. Si el número es múltiplo de 7, el producto se considera defectuoso y se devuelve al buzón de reproceso. En caso contrario, el producto se deposita, siempre que la meta de producción no se haya alcanzado.

Si la meta ya se ha cumplido, los productos que llegan desde el buzón de revisión son descartados y no se almacenan en el depósito. En este escenario, los operarios de calidad generan nuevos productos con el mensaje FIN y los envían al buzón de reproceso. Esto permite que, cuando un productor lea un producto con el mensaje FIN, detenga la producción.

Finalmente, una vez que la producción se ha detenido, los operarios de calidad continúan revisando los productos en el buzón de revisión, pero sin almacenarlos en el depósito.

```
//Esperamos a que todos los operarios terminen
for (int i = 0; i < numOperarios; i++) {
    productores[i].join();
    calidad[i].join();
}

sc.close();

System.out.println(x:"Proceso terminado.");
System.out.println("Numero de productos en el buzon de reproceso: " + reproceso.getCapacidad());
System.out.println("Numero de productos en el buzon de revision: " + revision.getCapacidad());
System.out.println("Numero de productos en el deposito: " + deposito.getCapacidad());
```

Figura 8: Finalización de ejecución

Para finalizar, una vez que todos los operarios han completado la ejecución de su método run, la clase Main espera a que todos los hilos terminen su proceso. Una vez finalizados, se imprime en la consola un mensaje indicando que el proceso ha concluido. Además, se muestra al usuario la cantidad de productos restantes en cada buzón.