Santiago Vela - 202026767

Verónica Escobar – 201922197

Caso 1: Manejo de Concurrencia

Infraestructura Computacional

2022-1

Contenido

[1. Introducción 1](#_Toc96256267)

[2. Diagrama de solución 1](#_Toc96256268)

[3. Funcionamiento global del Sistema 1](#_Toc96256269)

[3.1 Transmisión de mensajes 1](#_Toc96256270)

[3.2 Perspectiva de un buffer 2](#_Toc96256271)

[3.3 Sincronización entre threads 2](#_Toc96256272)

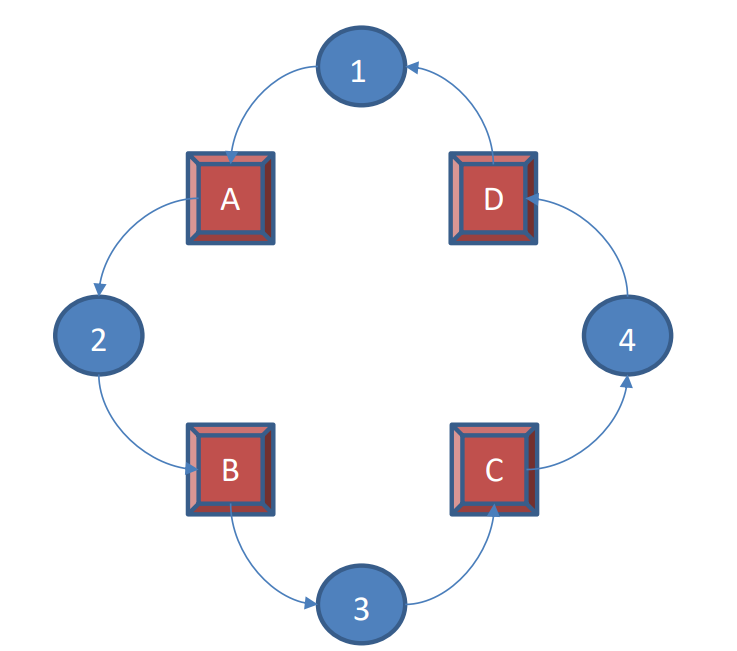
[3.4 Caso especial: Proceso 1 2](#_Toc96256273)

[3.5 Caso especial: Ultimo mensaje (FIN) 2](#_Toc96256274)

[4. Pruebas de funcionamiento 2](#_Toc96256275)

# Introducción

En el presente reporte se documenta el desarrollo e implementación de la simulación de un sistema de mensajes mediante el uso de Threads en un programa en Java.



El problema a solucionar consiste en encontrar una manera que en un sistema con 4 buzones y 4 procesos (threads) que conectan entre si a cada uno de los buzones en un patrón circular (ver figura 1). Cuando se inicia la ejecución el proceso 1 debe mandar el número fijo de mensajes, el cual fue dado por parámetro, al buzón que le sigue a ese thread (en este caso el A). Ya cuando un mensaje está dentro de un buzón este es procesado, y se le añade al final de este un mensaje que indica que ya paso por este buzón. Después, otro thread le pide al buzón sacar el mensaje y decide procesarlo, y al final del mensaje se le añade un mensaje que indica que paso por ese thread y se manda a dormir en thread un tiempo determinado. Eventualmente ese thread/proceso va a pasar el mensaje modificado al siguiente buzón, y el proceso continuara hasta que el mensaje vuelva a llegar al proceso 1. Este proceso se realiza para el número dado de mensajes que se desea dar, y una vez todos los mensajes hayan vuelto a llegar a el proceso 1 se decide mandar un último mensaje llamado fin que es el responsable de matar a todos los thread y terminar la ejecución del sistema de mensajería.

# Diagrama de solución

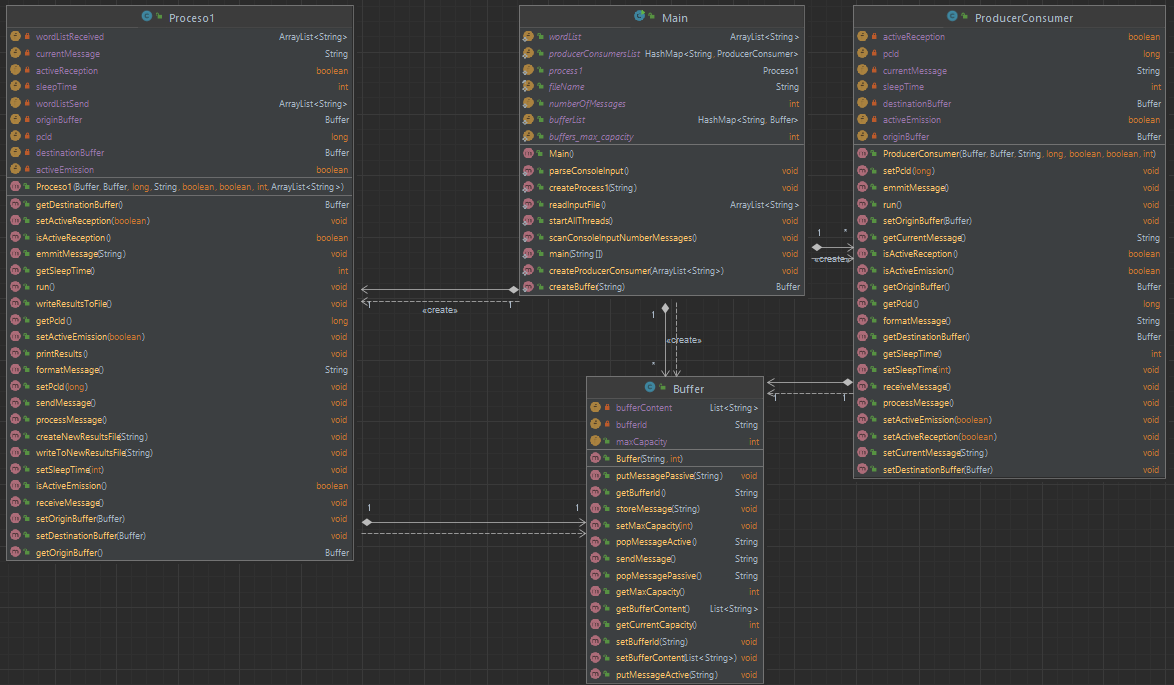
Para la creación del sistema de mensajería se implementaron 3 clases: Main, ProducerConsumer y Buffer.

En la Main es la responsable de leer la configuración de los buzones (buffers) y los procesos (producerConsumer) e inicializarlos, eventualmente esta misma es responsable de inicializar los threads y autorizar que el proceso 1 sea quien manda el número de mensajes dados por parámetro.

La clase de ProducerConsumer es una mezcla de las clases base de Producer y Consumer del ejemplo dado en clase del problema de Productores y Consumidores. Esta clase tiene la responsabilidad de sacar el mensaje del buzón de origen, procesarlo y eventualmente mandarlo al siguiente buzón.

La clase de Process1 es una especialización de la clase de ProducerConsumer (extiende producer consumer) para tratar con el caso especifico de procesos 1 , el cual tiene la particularidad que es la clase que manda todos los mensajes iniciales y es responsable de recibir al final los mensajes una vez le dan la vuelta a todos los buffers. Se necesito esta especialización dado que se necesita incluir parámetros nuevos como messagesToSend y una estructura del run que difiere de los otros procesos. La estructura del run en este caso consiste en primero mandar el numero de mensajes especificados al buffer A, y después prepararse para recibir el numero especificado de mensajes del buffer D, una vez se completa este ciclo este también tiene que mandar el mensaje de fin para finalizar a todos los otros procesos (del 2 al 4).

La última clase del modelo es buffer, es en esta clase que se implementa la mayoría de la sincronización del programa, ya que es aquí que se implementa las funcionalidades de intentar insertar un mensaje (putMessage) y de intentar sacar un mensaje (popMessage) los cuales los threads (ProducerConsumer) que conectan con los buzones ejecutaran para pasar los mensajes. Adicionalmente, buffer tiene métodos específicos par a implementar comunicación activa y pasiva, por eso hay dos versiones de putMessage y popMessage, cada una con el tipo de comunicación que utiliza.



# Funcionamiento global del Sistema

Inicialmente pensamos solo en crear una clase general para todos los Threads( procesos) en donde cada Thread compartía los mismos atributos y los mismos métodos, sin embargo en el momento de la ejecución el orden de instrucciones cambiaba, ya que el Thread 1 empieza generando o enviando mensajes al buffer A, mas no recibiendo es por eso que creamos una clase “Proceso1 (Thread1)”, en donde por parámetro le entra una lista de palabras previamente digitalizadas por sistema, y este mismo lo que hace es enviarlas al buffer “A” dependiendo su estado previamente seleccionado (Activo/Pasivo) y también dependiendo el almacenamiento del buffer, después de enviar las palabras por ultimo se le envía al Buffer un mensaje diciendo “Fin” .

De la misma manera el buffer A y, al igual que los demás Buffers, almacena los mensajes dependiendo su capacidad previamente instaurada, para que después el Thread siguiente (2) de manera pasiva o activa le solicite un mensaje al buffer. Cabe agregar que los Buffer no mandan mensajes y no retiran mensajes a menos que el Thread se lo solicite. Cuando se solicita un mensaje al Buffer y este no tiene nada adentro de el usamos “.wait()” o “.yield”. Cuando usamos “.wait()” lo que haremos será esperar a que algún mensaje le llegue al Buffer, o bien que se le pueda mandar a este algún mensaje, esto causara que se consuma RAM debido a que no realizara otras operaciones hasta que pueda tener un mensaje. Igualmente con el “yield” se realizara una espera activa, lo que significa que una vez se llegue a esta instrucción estará constantemente preguntando si ya hay mensajes debido a que se mandara a una cola de espera y este al ser único Thread se repetirá hasta que consuma un mensaje. Este mismo proceso entre el Thread y el buffer se hace secuencialmente hasta que llegue al Buffer D.

Al llegar los mensajes al Buffer D el Thread 1 de manera pasiva o activa estará solicitando los mensajes, y este una vez que tiene los mensajes los imprime por consola para después ser verificados, cabe agregar que cada vez que un mensaje pasa por un Thread este lo maraca con un símbolo distintivo el cual es el numero del Thread y si llego y se mando de manera pasiva o activa.

## 3.1 Transmisión de mensajes

* Recibir mensaje: esta función es única para todos los Threads, sin embargo dependiendo como se instaure el Thread, este pedirá mensajes de manera activa al Buffer usando la función “.yield()” la cual una vez implementada estará de manera constantemente solicitando mensajes al buffer . de igual forma podrá ser de manera
* Procesar mensaje: Una vez el Thread tiene el mensaje se tiene que identificar que el mensaje que se recibió no es “Fin”, una vez hecho esto se le tiene que agregar al mensaje el numero del Thread en el que esta actualmente y como ese mensaje llego ahí bien sea de manera activa o de manera pasiva.
* Emitir mensaje: finalmente el Thread enviara el mensaje ya marcado al buffer, sin embargo este envió dependerá del almacenamiento del buffer y de igual forma si su envió es de forma activa o pasiva.

## 3.2 Perspectiva de un buffer

## 3.3 Sincronización entre threads

## Caso especial: Proceso 1

Como se menciono anterior mente El proceso 1 o Thread 1 contiene los mismos métodos que la clase “Producerconsumer” sin embargo su proceso de ejecución es diferente, ya que este al recibir una lista por entrada tendrá que enviar los mensajes al Buffer A y no como los demás que simplemente solicitan y después envían.

## Caso especial: Ultimo mensaje (FIN)

Este mensaje que representa el final de la lista de mensajes, es enviado por la clase “Proceso1” la cual le indica a los Thread que ese mensaje no se marca, y que de igual forma no vendrán mas mensajes después del mismo. Este mensaje se manda después de enviados todos los mensajes por parte de la lista.

# Pruebas de funcionamiento