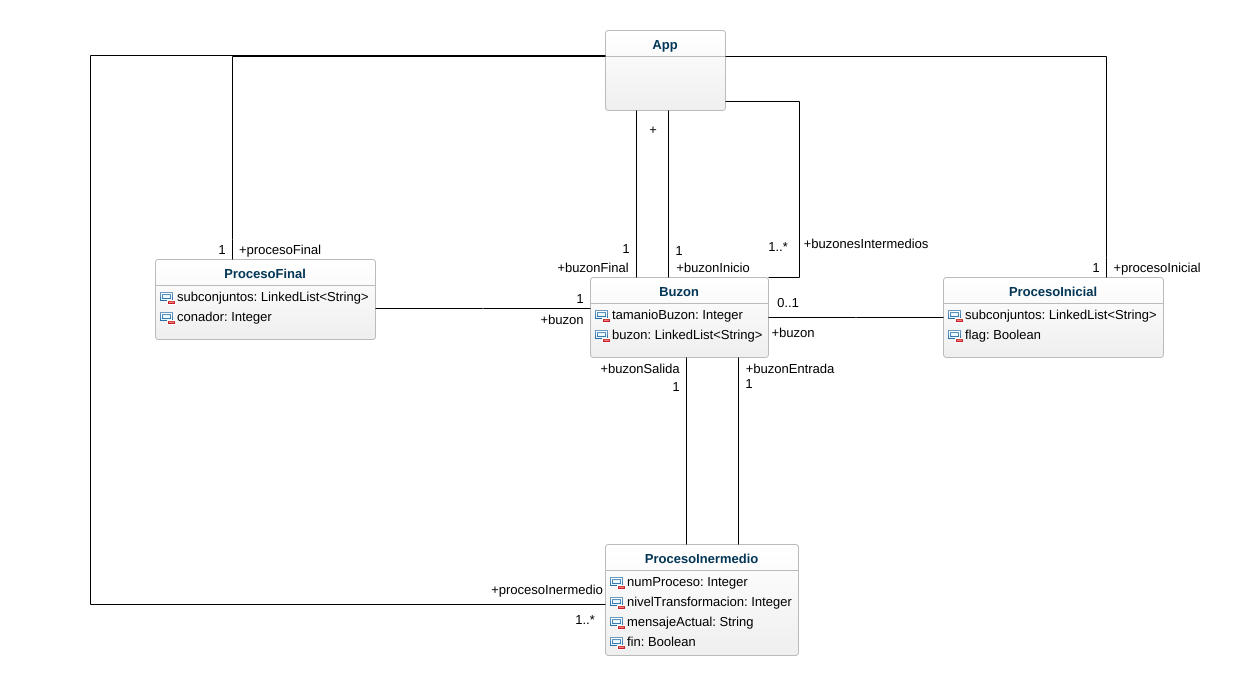
**Caso 1 Infraestructura Computacional**

Andrés Molina *a.molinam* - 201923434

Pablo Guatibonza *p.guatibonza* - 202014393

Diana Silva *da.silvaa* - 201815366

1. Diagrama de clases

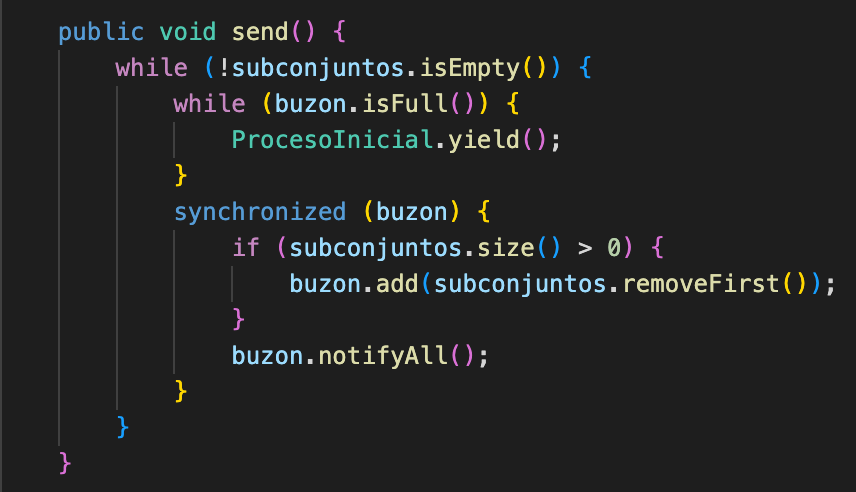


1. Funcionamiento del programa

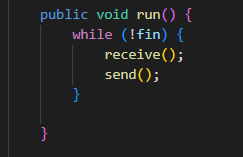
Para entrar a analizar el funcionamiento del caso, explicaremos el porqué de las decisiones que se tomaron en cuanto a las clases creadas. En la clase App se encuentra el método Main, en él creamos la ejecución del programa, así es como instanciamos la relación entre los threads y los buzones. Encontramos hasta cierto punto necesario más por practicidad y orden en el código separar el proceso inicial, el final y el intermedio, teniendo presente que los tres funcionan como threads. Una alternativa que se puede tomar, por atributos que estas clases comparten es crear una clase contenedor con los atributos compartidos y heredar de ella los procesos que se ejecutarán, pero la manera en que se decidió ayuda a entender un poco mejor la arquitectura global en su funcionamiento.

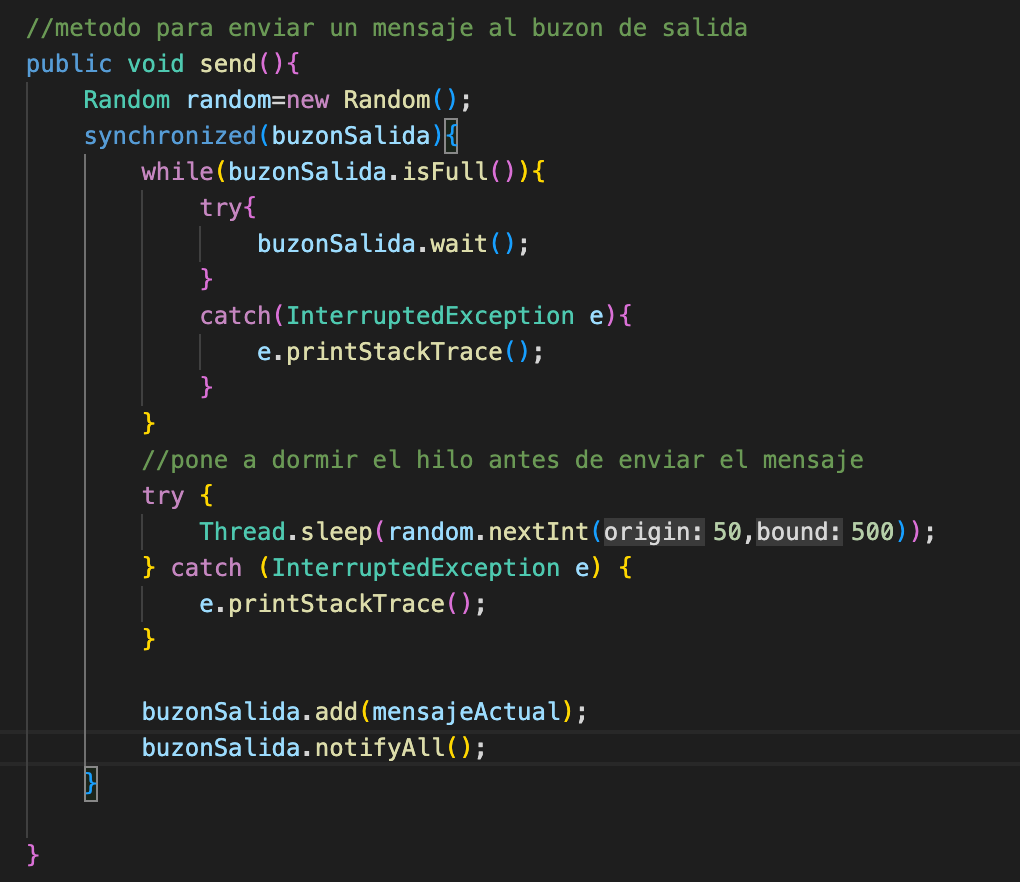
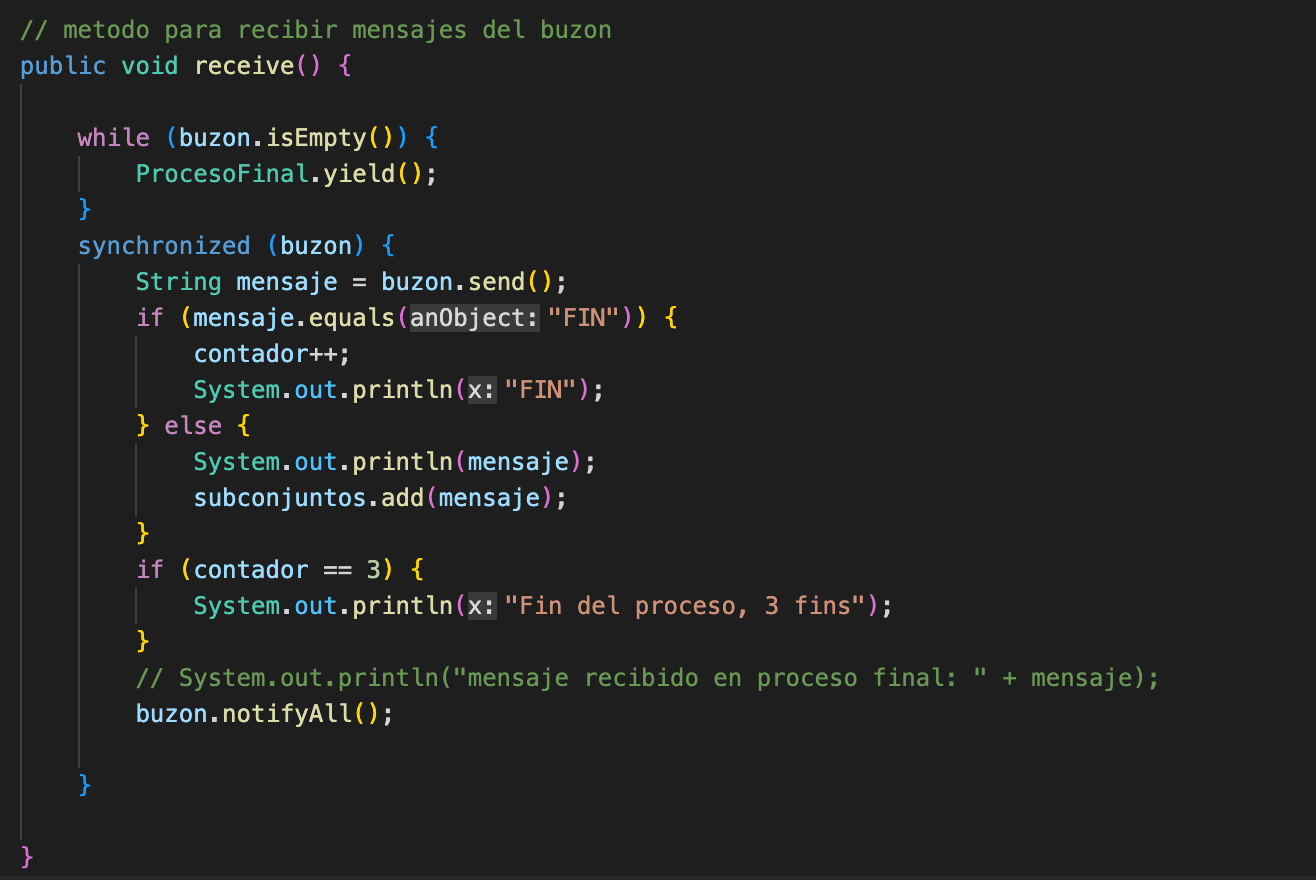
Teniendo en mente el porqué de la arquitectura de clases, entramos a analizar las decisiones que se tomaron frente a la interacción entre threads y buzones, empezaremos por explicar la relación entre el *proceso inicial y el buzón*

* Proceso inicial – Buzón: Según es mencionado en la documentación del caso, necesitamos que este primer thread divida la n cantidad de subconjuntos que se desee y proceda a almacenarlos en el buzón, esto se lleva a cabo en el método *send()* de la clase *ProcesoInicial* que se ejecuta a través del método *run*() que es invocado por el main al llamar al *<thread>.start()*,adicional a esto, debemos tener en cuenta que los buzones tienen una capacidad máxima, así que en el momento de dividir los subconjuntos se debe verificar que exista espacio en el buzón. En caso de no haberlo, se procede a esperar hasta que exista el espacio para almacenar un subconjunto, este proceso lo realizamos mediante espera activa, a través del método *yield(),* este método nos permite mantener activo el thread en la espera de espacio en el buzón. Una vez se almacena el subconjunto en el buzón se pasará a la siguiente fase del programa y se hace un llamado a los threads intermedios al saber que hay algo nuevo almacenado en el buzón. También, podemos mencionar que el procesoInicial almacena todos los subconjuntos, además de tres mensajes de FIN, para que cuando lleguen a los mensajes intermedios estos sepan que su proceso culminó.



* Procesos intermedios - Buzón: En este momento de la ejecución del programa, debemos tener presente que, según la documentación del caso, hay que realizar una transformación a cada subconjunto al hacer su paso por un proceso, completando al final 3 transformaciones por cada subconjunto. Nos remitiremos entonces a la clase *ProcesoIntermedio* que a diferencia del proceso anterior debe también recibir un dato, lo que vendría siendo extraerlo del buzón, este proceso se realiza en el método *receive().*  En este método hacemos una espera pasiva esperando a que el buzón contenga algo que el proceso pueda retirar, esto se realiza mediante *wait().*   
  Una vez podemos recibir el subconjunto de un buzón se tienen que hacer un par de validaciones, pues debemos verificar que el subconjunto que recibimos sea diferente a ‘FIN’ para poder enviarlo al siguiente buzón con su respectiva transformación. Pero, si es el subconjunto final (denotado por la cadena de caracteres ‘*FIN’*) se cambia el estado del programa para determinar el final de este subconjunto, es decir, el proceso intermedio se ejecuta mientras no haya llegado el mensaje FIN, si llega este mensaje se termina el método *run().*  
  Una vez se recibe del buzón un subconjunto se notifica a los threads para que puedan almacenar un subconjunto nuevo en el buzón.  
  

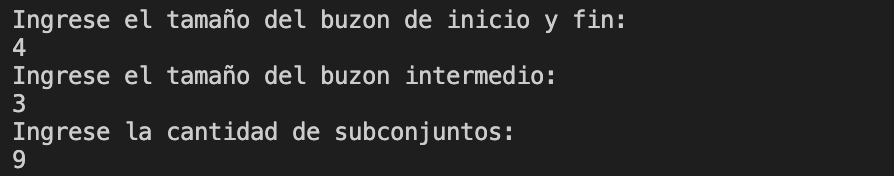


* Estructura del mensaje:  
    
    
  El mensaje inicial(M7) es almacenado por el proceso Inicial. En cada proceso es transformado según el nivel de transformación. En primer lugar, pasa por el proceso de nivel 1 con nivel de transformación 0, para que el mensaje quede así: M7T01. Posteriormente, pasa por el proceso de nivel 1 con nivel de transformación 1, para que el mensaje quede M7T01T11. Finalmente, el mensaje pasa por el proceso de nivel 1 con nivel de transformación 2, para que el mensaje quede M7T01T11T21.   
    
  Ahora explicaremos el método de envío al siguiente buzón, esto se realiza mediante el método *send()* de esta misma clase. Debemos entonces hacer la transformación del subconjunto que recibimos del buzón y, para poder almacenarlo debemos generar una espera de entre 50 y 500 milisegundos. Una vez el buzón de envío se encuentre con espacio disponible se almacenará en él, en caso contrario se espera pasivamente a que este se encuentre con espacio disponible.  
  Estos dos métodos se ejecutan en el método *run()* en el thread mientras no se llegue al mensaje final.
* Buzón - Proceso final: Una vez el subconjunto llega al proceso final, este debe verificar la llegada final de los subconjuntos, esto se hará mediante el método *receive()* de la clase *ProcesoFinal,* es entonces cuando se hace la verificación de llegada de 3 mensajes de final, uno por cada hilo de procesos, y cuando se terminen de procesar los subconjuntos y lleguen los mensajes de fin, se sabrá que la ejecución del programa llegará a su final.  
  Por último, se deben imprimir las transformaciones de los subconjuntos una vez atraviesan todos los procesos al igual que los 3 mensajes de FIN, esto se mostrará por consola.

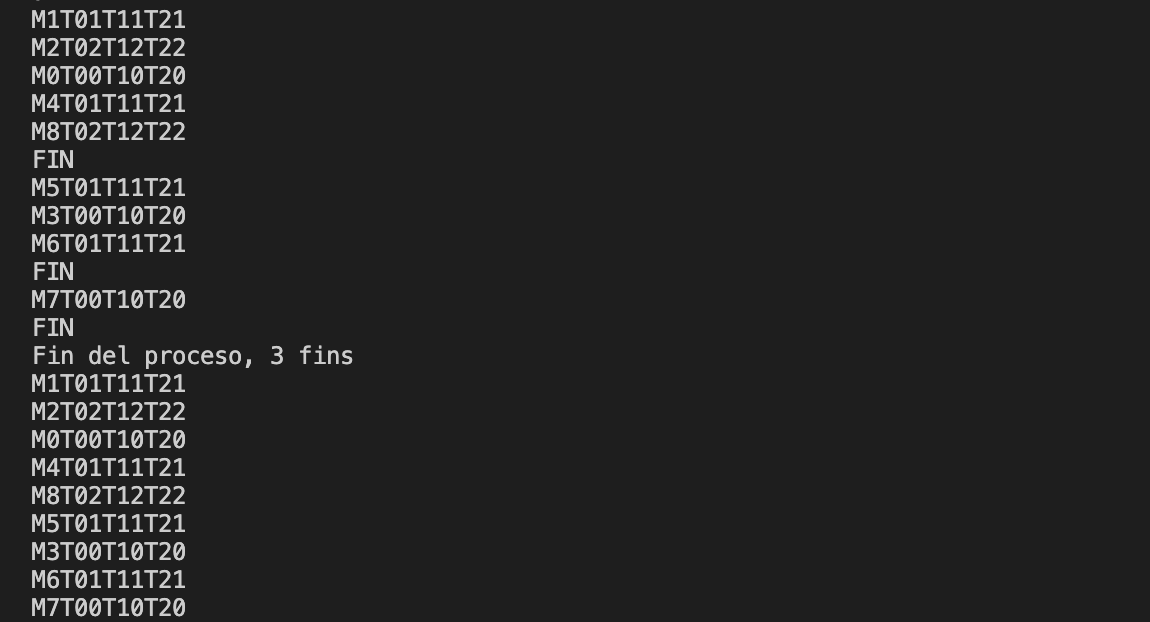
1. Validación

Al correr el programa, este nos preguntara por el tamaño del buzón inicial y el final, además de preguntarnos por la cantidad de subconjuntos con los que contaremos, a continuación, una demostración del funcionamiento por consola del programa.

Usaremos como ejemplo un buzón inicial y final de capacidad 4, buzón intermedio de capacidad 3 y una cantidad de subconjuntos igual a 9:



Una vez ejecutada la última instrucción el programa comenzará a correr respetando los tiempos de espera de los buzones. Al final el output será:



* Funcionamiento de la sincronización

El proceso inicial alberga todos los subconjuntos que se deben enviar al buzón respectivo. Para hacer este envío, es necesario tomar el buzón como monitor y no cederlo hasta que se terminen los subconjuntos, o hasta que el buzón esté lleno. En este último caso, el procesador inicial cede el thread de manera activa, es decir, cede el proceso, pero lo vuelve a pedir inmediatamente. Si se terminan los subconjuntos, solamente se termina el proceso y se avanza al siguiente paso.

Cuando se cede el procesador que tiene como monitor al buzón inicial, los otros 3 procesos del nivel de transformación 0 compiten para ver quién se queda primero con este. El primero que lo coja retira el mensaje del buzón y le hace su transformación respectiva. Posteriormente, se notifica a los otros threads que están esperando el procesador y vuelven a competir por tomar el monitor.

Entre procesos intermedios (de una misma fila) funciona de una manera similar la comunicación. Se toma el monitor sobre el buzón de salida para depositar el mensaje que se acabó de transformar. Si el buzón está lleno, entonces el proceso entra en espera pasiva mientras el buzón deja de estar lleno. Si no está lleno se pone a dormir el thread entre 50 y 500 ms para luego enviar el mensaje. Cuando el mensaje es enviado se notifica a los procesos que están esperando (siguiente nivel de transformación) y se cede el procesador. El proceso del siguiente nivel de transformación toma el procesador y saca el mensaje siempre y cuando el buzón no esté vacío. Si lo está, espera pasivamente. Si no, lo saca, cede su procesador y notifica a los procesos en espera.

Lo mismo sucede con el proceso final. Saca mensajes de su buzón respectivo siempre y cuando no esté vacío. Si lo está, espera activamente, con el método *yield* del thread. Si no lo está toma el procesador, saca el mensaje y notifica a los procesos que estén esperando.