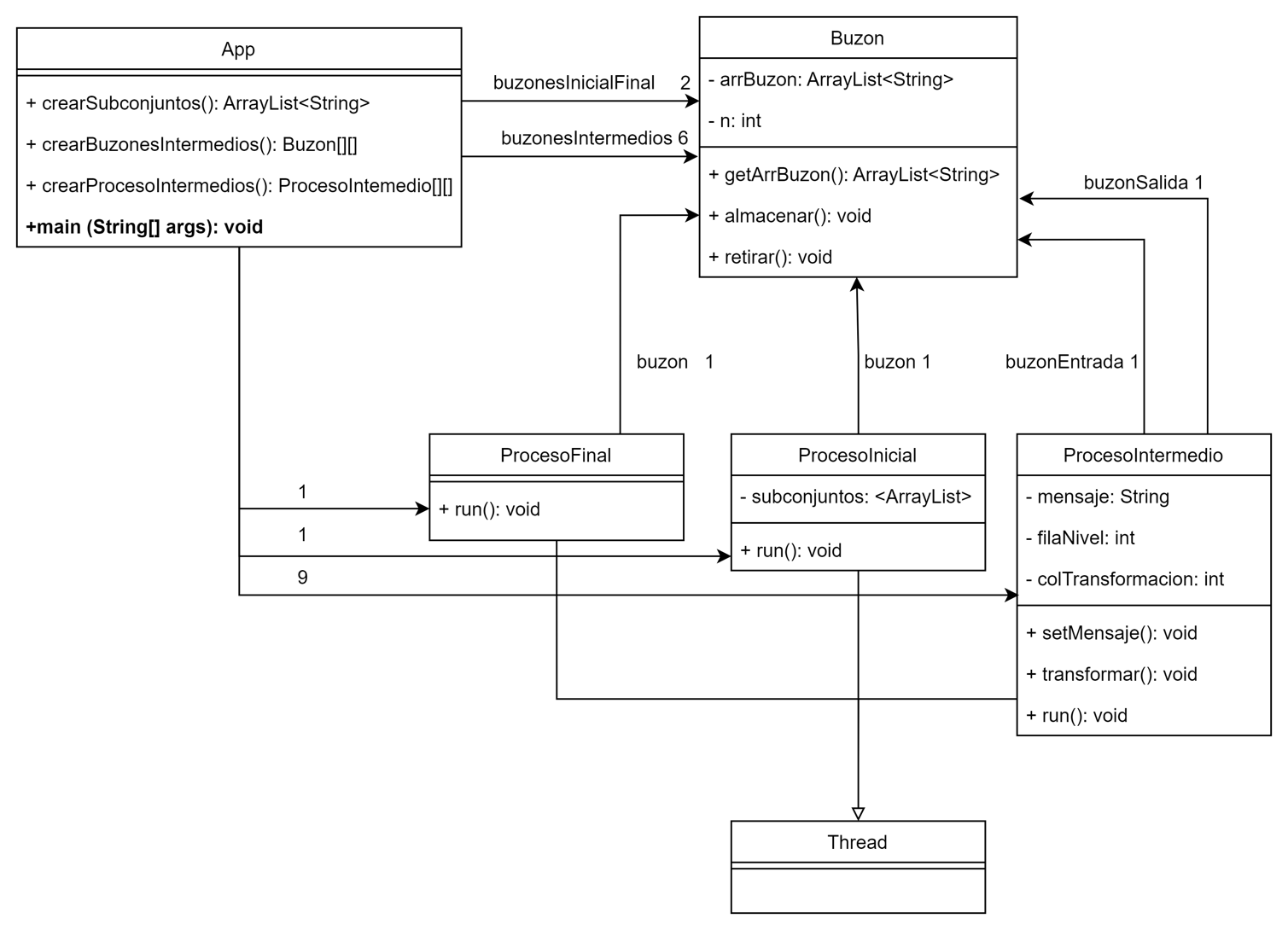
**Caso 1 – Infraestructura Computacional**

Mariana Ruiz - 202011140

David Alejandro Ruiz - 201620095

Nicolás Klopstock – 202021352

**Diagrama de clases:**



El modelo de solución cuenta con 5 clases, una clase principal (*App)*, una clase que cumple la función de Buffer (*Buzon)* y tres clases que representan los procesos los cuales extienden de la clase Thread de java (*ProcesoInicial, ProcesoFinal, ProcesoIntermedio)*.

La clase *App*, se encarga de iniciar el programa (método main) pidiéndole al usuario el tamaño de los buzones de inicio, fin e intermedios, así como la cantidad de subconjuntos (mensajes) a procesar. A continuación, el método main crea los subconjuntos y los procesos con los buzones correspondientes. Finalmente, en el método main se inician los procesos.

Se estableció las clases *ProcesoInicial* y *ProcesoFinal,* cada una con una relación unidireccional a la clase buzón. Para *ProcesoInicial*, representa el buzón de inicio al cual el proceso comienza a enviar los subconjuntos de manera semiactiva. Para *ProcesoFinal*, representa el buzón de salida del cual el proceso retira los subconjuntos transformados; este proceso se ejecuta de manera semiactiva.

Por otra parte, se definió la clase *ProcesoIntermedio* que cuenta con dos relaciones unidireccionales, la primera representa el buzón de donde se retiran mensajes y la segunda representa el buzón donde se almacenan los subconjuntos transformados; este proceso se ejecuta de manera pasiva.

Finalmente, se decidió implementar la clase *Buzon* para comunicar procesos y estos puedan pasarse información.

**Funcionamiento del programa:**

El programa funciona de la siguiente manera: En la clase App, se le pide al usuario el tamaño de los diferentes buzones y el número de subconjuntos que quiere mandar y se crean los subconjuntos (Estos incluyen los mensajes de FIN), buzones y procesos; logrando que estos dos últimos “simulen” una matriz (como en el ejemplo del enunciado). En el caso de los Buzones, una matriz 3x2 y en el caso de los procesos una 3x3. El método main da inicio a todos los procesos.

Para que un mensaje llegue de ProcesoInicial a ProcesoFinal se ejecuta el método run() de la clase ProcesoInicial. Este almacena los subconjuntos (mensajes) en el buzón inicial llamando el método almacenar() de la clase Buzon. Al ser el primer almacenamiento, no necesitamos tener en cuenta el nivel de transformación ni la fila de procesos intermedio por la que pasó. Aquí se presenta un momento de sincronización entre ProcesoInicial y Buzon, ya que ProcesoInicial debe verificar si el buzón tiene espacio para poder poner un mensaje. En caso de que el buzón inicial no tenga espacio, se utiliza yield() para manejar una espera semiactiva.

Luego, un proceso intermedio llama la función retirar() de Buzon para retirar un mensaje de su respectivo buzón de entrada. Aquí se presenta un segundo momento de sincronización, esta vez entre ProcesoIntermedio y Buzon. Esto se da, dado que cada Thread debe verificar si el buzon de entrada tiene elementos para retirar y dos procesos no pueden retirar el mismo elemento. Lo mismo va a suceder justo después, cuando el proceso intermedio intente almacenar en el siguiente buzón algún mensaje con el método almacenar() de la clase Buzon.

Cuando se llegue a la última transformación realizada por los procesos intermedios, el último de estos debe almacenar el mensaje actual en el buzón final. Acá se presenta la misma situación de almacenamiento descrita anteriormente. Cuando el proceso intermedio termina de almacenar, el proceso final empieza a retirar usando la misma función retirar(). Aquí se presenta una situación de sincronización, pero esta vez en entre las clases ProcesoFinal y Buzon. Cada que retira un subconjunto, representa el final de la vida de un Thread; entonces imprime el mensaje final generado y acaba ese proceso. Este último proceso también se realiza por medio de espera semi activa, como en el caso del ProcesoInicial.

Cabe aclarar que, si bien, en ProcesoInicial se utiliza el método almacenar de Buzón que implementa una espera pasiva, realmente, la espera es semiactiva. La condición para que ProcesoInicial haga yield() es que el buzón esté lleno. Si no lo está, se llama al método almacenar() que hace esta misma comprobación. Así que en ningún momento va a cumplir la condición de estar lleno porque ya se verificó con anterioridad dentro del run de ProcesoInicial y no va a ejecutar el wait(). Esta misma situación ocurre en el caso del ProcesoFinal pero con el método retirar().

**Validaciones:**

En todos los casos probados la ejecución termina exitosamente. Inicialmente se probó con cantidades de subconjuntos y tamaños de los buzones pequeños. Por ejemplo, de 10 subconjuntos, tamaño de buzones iniciales y finales de 6 y tamaño de buzones intermedios de 4. Posteriormente se hicieron pruebas con valores más grandes. Por ejemplo, de 1000 subconjuntos, tamaño de buzones iniciales y finales de 50 y tamaño de buzones intermedios de 30. Aunque en esta última prueba el programa se demora un tiempo mayor a 30 segundos, termina exitosamente.

Se envían mensajes cada vez que el ProcesoInicial envía un mensaje, cada vez que un ProcesoIntermedio transforma un mensaje (con el tiempo en que se demoró esta transformación), cada vez que un ProcesoIntermedio almacena un mensaje y cada vez que el ProcesoFinal retira un mensaje. Se decidió no incluir cada vez que un ProcesoIntermedio retira un mensaje porque hacía más difícil el entendimiento de los mensajes.

Debido a que no se envían mensajes cada vez que un ProcesoIntermedio retira un mensaje, en la parte inicial hay casos en donde parece que ProcesoInicial está almacenando más mensajes que lo que permite su capacidad. Sin embargo, esto es porque en medio de estos mensajes de almacenamiento inicial, algún ProcesoIntermedio retiró un mensaje del buzón principal y esto no es notificado en consola. Si se quiere verificar que esto no está ocurriendo, puede quitar el comentario de la línea 67 en ProcesoIntermedio.