**Caso 3 -** Concurrencia y Sincronización de Procesos

**Grupo 10**

Sergio Soler Garzon - 202310103

Daniel Diab: 202321332

**Tecnología e Infraestructura Computacional**

Ricardo Gómez

**Universidad de los Andes**

**Ingeniería de Sistemas y Computación**

**Bogotá D.C.**

**2025**

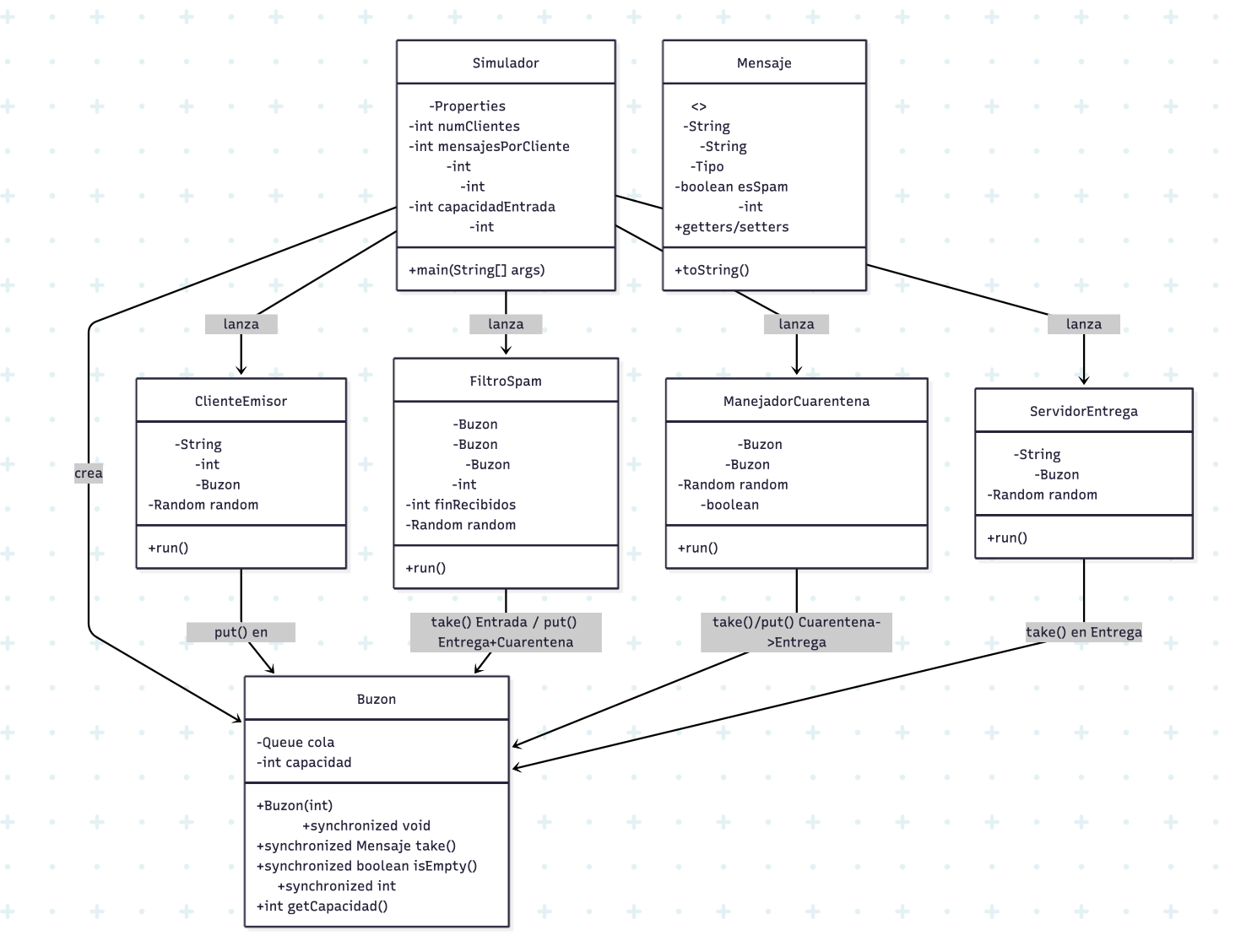
1. ***Descrpcion general del sistema***

El sistema simula un centro de mensajería distribuido con varios actores concurrentes: clientes emisores, filtros de spam, manejador de cuarentena y servidores de entrega. Cada componente opera en hilos independientes, intercambiando mensajes mediante buzones sincronizados con las primitivas básicas de Java (synchronized, wait, notify, notifyAll). Ademas, el sistema simula un centro de mensajería con cuatro actores concurrentes:

* Clientes emisores: producen mensajes con cabeceras de INICIO y FIN, y N mensajes normales.
* Filtros de spam: consumen del buzón de entrada, clasifican y enrutan a entrega o cuarentena.
* Manejador de cuarentena: decrementa el “tiempo de cuarentena”, descarta algunos mensajes maliciosos y libera el resto hacia entrega.
* Servidores de entrega: consumen de entrega y procesan hasta recibir FIN.

1. ***Diagrama de clases***

El siguiente diagrama muestra las principales clases y sus relaciones.



1. ***Sincronizacion por pares de objetos***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pareja** | **Recurso compartido** | **Riesgo** | **Mecanismo implementado** | **Funcionamiento** |
| ClienteEmisor y Buzon(Entrada) | *cola* del buzón de entrada (capacidad limitada). | Overflow si varios productores agregan simultáneamente; busy-wait ineficiente. | *synchronized* en *put*, condición *while* (*size* == *capacidad*) *wait(),* *notifyAll()* al agregar. | Garantiza exclusión mutua y suspensión pasiva cuando se llena; se reanuda cuando un consumidor libera espacio. |
| FiltroSpam y Buzon (Entrada/Entrega/ Cuarentena) | Colas de entrada y de salida. | Inconsistencias por accesos concurrentes; pérdida de mensajes. | *synchronized* en *take*/*put* de cada buzón; espera pasiva en *take* | Nunca se “pierde” un mensaje: *take* bloquea cuando está vacío y *put* despierta consumidores. |
| ManejadorCuarentena y Buzon (Cuarentena/Entrega) | Cola de cuarentena y de entrega. | Lecturas “a medio actualizar”; carrera entre decrementos y envíos. | El manejador extrae con *take* (bloqueante cuando haya), mantiene lista local *enRevision*, y transfiere a Entrega con *put.* | Separar “lectura” (del buzón) de “cómputo” (lista local) evita bloquear la cola de cuarentena por largos periodos. |
| ServidorEntrega y Buzon(Entrega) | Cola de entrega (varios consumidores). | Hambre o consumo exclusivo del FIN por un solo servidor. | *take* bloqueante con *synchronized* y *wait*/*notifyAll.* | Cada servidor avanza uno a uno sin condiciones de carrera; (ver “Decisiones y discrepancias”). |
| Filtros y ManejadorCuarentena (vía Buzón) | Flujo de mensajes a cuarentena y señalización de fin. | Terminar antes de vaciar cuarentena. | Filtros envían mensajes spam con tiempo; el manejador decrementa y transfiere o descarta. | Flujo acotado: cada ciclo (1 s) avanza tiempos y decide; no hay precondición circular, no deadlock. |

1. ***Funcionamiento global*** 
   1. *Clientes*: Depositan INI, luego N NORMAL, y finalmente FIN en Entrada.
   2. *Filtros:* Consumen de Entrada:
   * INI: redirigen a Entrega (sirve de marca de inicio global).
   * NORMAL: si *esSpam*, lo envían a Cuarentena con *tiempoCuarentena* ∈ *[10000,20000]* ms; si no, lo envían a Entrega.
   * FIN de cliente: acumulan conteo interno y también lo redirigen a Cuarentena para que el manejador conozca el fin.
   * Cuando detectan que terminaron todos los clientes, envían un FIN de sistema a Entrega (ver nota en §7).
   1. Manejador de cuarentena (ciclo cada ~1 s):
      * Decrementa *tiempoCuarentena* de cada mensaje en revisión.
      * Al llegar a 0: descarta con prob. ≈ 1/7 (múltiplos de 7) o transfiere a Entrega; termina al recibir un FIN.
   2. *Servidores de entrega:* Procesan uno a uno hasta recibir FIN.
2. ***Decisiones de diseño***

Cantidad de mensajes por cliente: de acuerdo con el enunciado nos sugiere aleatorio entre 20 y 100; el simulador usa mensajes desde config.txt.Para alinearlo sin tocar la arquitectura se debe generar numMensajes = 20 + rand(81) en ClienteEmisor si mensajes == -1 (p. ej.) y documentarlo.En cuando a cuándo enviar FIN a Entrega: Se sugiere hacerlo cuando Entrada y Cuarentena estén vacíos y no habrá nuevos mensajes, y luego copiar FIN para todos los servidores. *Actual:* el filtro envía FIN-ENTREGA al completar finRecibidos == totalClientes. *Cierre simple:* antes de poner FIN-ENTREGA, verificar: entrada.size()==0 && cuarentena.size()==0. Después de poner un FIN, poner copias adicionales hasta que cada servidor lo reciba (o publicar numServidores FIN). Debido a que garantiza terminación ordenada y que todos los servidores concluyan. Con respecto a los servidores en “espera activa” de acuerdo al enunciado del caso: se podría implementar como un lazo con poll() + Thread.yield()/sleep(…). Esto porque se prefirió espera pasiva (take() bloqueante) para no malgastar CPU y mantener el sistema justo y estable.y el número de filtros: en Simulador, el for de filtros usa < numFiltros (no <=): lanza numFiltros-1 filtros. El cierre se realiza al cambiar a for (int i=1; i<=numFiltros; i++) ….

1. ***Análisis de concurrencia (riesgos y cómo se previenen)***

Todas las operaciones sobre las colas de los buzones están protegidas por bloques synchronized, y el uso conjunto de las condiciones *while + wait()* garantiza que no se exceda la capacidad ni se intente leer cuando el buzón está vacío, evitando así condiciones de carrera. No existe pérdida de mensajes porque todo el flujo, desde la entrada hasta la entrega o la cuarentena, se realiza mediante llamadas a *put()* y *take()* que son atómicas y ejecutadas bajo el mismo monitor. Tampoco se presentan interbloqueos, ya que no hay bloqueos anidados entre monitores; cada operación *put()* o *take()* accede a un solo monitor y no espera otros recursos, evitando ciclos de espera circular. Finalmente, no se produce hambre (starvation) porque el uso de *notifyAll()* despierta a todos los hilos en espera, garantizando que ninguno quede permanentemente bloqueado. Además, la disciplina FIFO de la estructura *LinkedList* utilizada en cada buzón favorece un progreso justo y equitativo entre los hilos productores y consumidores.

1. ***Plan de validación (pruebas realizadas)***

Ejecuta cada caso modificando config.txt. Los **resultados esperados** sirven como oráculo rápido.

* 1. *Prueba A, Flujo básico*
     + Parámetros: clientes = 3, mensajes = 5, filtros = 2, servidores = 2, capEntrada = 10, capEntrega = 10.
     + Esperado: Cada cliente imprime INI, 5 NORMAL, y FIN.
     + Los filtros muestran envíos a Entrega o Cuarentena coherentes con el flag esSpam.
     + El manejador imprime decrementos y transfiere o descarta; al final, vacía su lista y termina.
     + Los servidores procesan y reciben al menos un FIN, terminando limpiamente.
     + No quedan mensajes en los buzones (tamaños 0).
  2. *Prueba B, Presión de entrada (capacidad pequeña)*
     + Parámetros: capEntrada=1, resto como Prueba A.
     + Esperado: productores bloquean frecuentemente en put(); no hay pérdidas; el sistema drena.
  3. *Prueba C, Cuarentena cargada*
     + Parámetros: forzar tasa alta de spam (p. ej., en código o aumentando random.nextBoolean() hacia true).
     + Esperado: el Manejador mantiene muchos mensajes en revisión, reduce tiempos y libera de forma continua; algunos se descartan (múltiplos de 7).
  4. *Prueba D, Terminación limpia con múltiples servidores*
     + Parámetros: servidores = 4.
     + Esperado: todos los servidores terminan tras recibir FIN.
  5. *Métricas de chequeo rápido (logs):*
     + Conteo total: 3 clientes × (1 INI + 5 NORMAL + 1 FIN) = 21 eventos de cliente → deben verse ruteados y procesados (con algunos NORMAL desviados a cuarentena).
     + En la terminación: buzonEntrada.size()==0 && buzonEntrega.size()==0 && buzonCuarentena.size()==0.