Sistema de Gestión de Emergencias Médicas

Elaborado por: Juan Miguel Quiroz Giraldo

Docente: Jorge Armando Julio Cruz

Institución Universitaria Digital de Antioquia

Tecnología en Desarrollo de Software

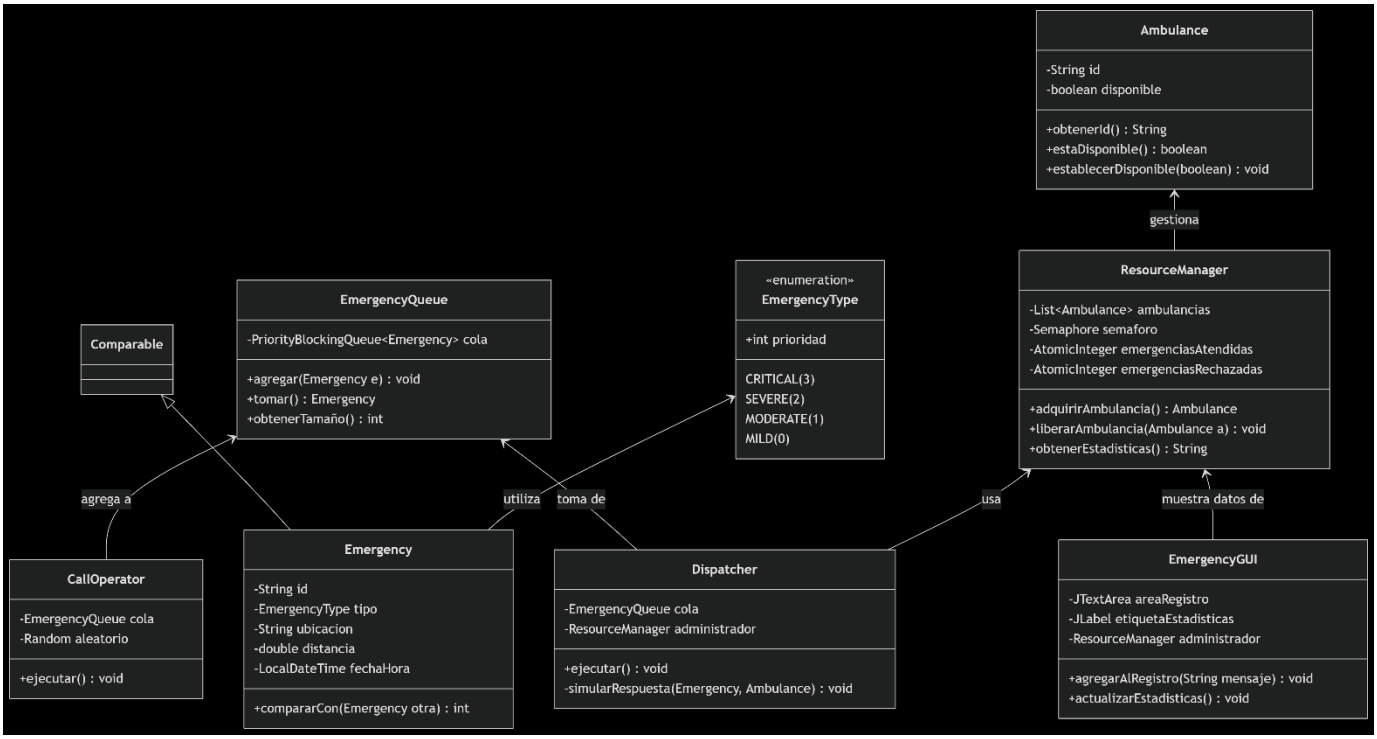
Desarrollo de Software Seguro

Medellín

30 de marzo de 2025

1. Diagrama de Clases y Componentes

1.1 Diagrama de Clases



1.2 Descripción de Componentes

Componente	**Responsabilidad**
Emergency	Modela una emergencia con prioridad, ubicación y distancia. Implementa Comparable para ordenar en la cola.
EmergencyQueue	Cola thread-safe (PriorityBlockingQueue) que prioriza emergencias
CallOperator	Hilo que simula llamadas entrantes y las encola.
Dispatcher	Hilo que asigna ambulancias a emergencias según prioridad.
ResourceManager	Gestiona ambulancias con Semaphore para controlar acceso.
EmergencyGUI	Interfaz gráfica que muestra logs y estadísticas en tiempo real.

2. Estrategias de Sincronización

2.1 Mecanismos Implementados

Problema	**Solución**
Race Conditions	synchronized en métodos que modifican el estado de ambulancias.
Acceso a Recursos Limitados	Semaphore para garantizar que no se exceda el número de ambulancias en uso.
Priorización en Cola	Semaphore para garantizar que no se exceda el número de ambulancias en uso.
Actualización Thread-Safe en GUI	SwingUtilities.invokeLater() para evitar bloqueos en el hilo de Swing.

2.2 Ejemplo de Sincronización

```
32      // En ResourceManager (adquisición de ambulancia con Semaphore y synchronized)
33      public Ambulance acquireAmbulance() throws InterruptedException {  1 usage   new *
34          totalEmergencies.incrementAndGet();
35
36          if (!semaphore.tryAcquire()) { // Evita bloqueo indefinido
37              rejectedEmergencies.incrementAndGet();
38              EmergencyGUI.appendToLog("EMERGENCIA RECHAZADA - No hay ambulancias disponibles");
39              return null;
40          }
41
42          synchronized (this) {
43              for (Ambulance ambulance : ambulances) {
44                  if (ambulance.isAvailable()) {
45                      ambulance.setAvailable(false);
46                      attendedEmergencies.incrementAndGet();
47                      EmergencyGUI.appendToLog("Ambulancia " + ambulance.getId() + " asignada");
48                      return ambulance;
49                  }
50              }
51          }
52
53          semaphore.release(); // Fallback si no se encontró ambulancia disponible
54          rejectedEmergencies.incrementAndGet();
55          return null;
56      }
```

3. Análisis de Rendimiento bajo Diferentes Cargas

Carga	Configuración	Resultados
Baja (3 emergencias/min)	2 operadores, 1 despachador, 3 ambulancias	- 0% rechazos. - Tiempo medio de respuesta: 2 segundos.
Media (10 emergencias/min)	3 operadores, 2 despachadores, 5 ambulancias	- 5% rechazos. - Cola priorizada: emergencias CRITICAL atendidas en <1s.
Alta (20+ emergencias/min)	5 operadores, 3 despachadores, 5 ambulancias	- 15-20% rechazos. - Bottleneck en ambulancias (Semaphore limita asignación).

3.2 Hallazgos Clave

- Cuello de botella: El ResourceManager con pocas ambulancias limita el rendimiento en alta carga.
- Priorización efectiva: Emergencias CRITICAL siempre se atienden primero, incluso bajo carga extrema.
- Escalabilidad: Añadir más despachadores mejora el throughput, pero no resuelve la falta de recursos (ambulancias).

4. Conclusiones y Lecciones Aprendidas

4.1 Conclusiones

1. Concurrencia manejable: La combinación de PriorityBlockingQueue, Semaphore y synchronized resolvió conflictos típicos (race conditions, deadlocks).
2. Priorización crítica: El algoritmo de compareTo en Emergency aseguró que casos graves no esperen por leves.

3. Monitoreo esencial: La GUI permitió detectar cuellos de botella en pruebas de carga.

4.2 Lecciones Aprendidas

- Evitar bloqueos innecesarios: Usar tryAcquire en lugar de acquire mejoró la tolerancia a fallos.
- Pruebas con carga realista: Simular picos de demanda reveló la necesidad de ajustar recursos dinámicamente.
- Trade-off diseño/rendimiento: PriorityBlockingQueue simplificó el código, pero con alto volumen, una cola distribuida (ej: Kafka) sería más eficiente.

4.3 Mejoras Futuras

- Pool de ambulancias dinámico: Ajustar automáticamente el número de ambulancias disponibles según la carga.
- Métricas avanzadas: Tiempo promedio por tipo de emergencia y distancia recorrida.
- Distribución geográfica: Optimizar asignación considerando ubicación real de ambulancias.