



**udp** UNIVERSIDAD  
DIEGO PORTALES

ESCUELA DE INFORMÁTICA & TELECOMUNICACIONES

ESTRUCTURAS DE DATOS & ALGORITMOS

---

## Laboratorio n°4

---

**Autores:**

Allen Mora  
Gabriel Varas

**Profesor:**

Marcos Fantóval

**Fecha:**

09/06/2025

---

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Metodología</b>	<b>2</b>
<b>3. Experimentación y Resultados</b>	<b>11</b>
3.1. Generación de Pacientes . . . . .	11
3.2. Configuración de la Simulación . . . . .	12
3.3. Resultados . . . . .	12
<b>4. Conclusiones</b>	<b>13</b>

---

## 1. Introducción

En este laboratorio se diseñó un sistema de gestión de pacientes para el área de urgencias de un hospital. Esto se logró utilizando colas de prioridad, mapas y montones(heaps), dicha gestión se basó en la categorización oficial del Ministerio de Salud de Chile(C1 a C5).

## 2. Metodología

Para la gestión de cada paciente se utilizó la clasificación utilizada actualmente por el Ministerio de Salud de Chile, la cual cuenta con 5 categorías:

- C1: Emergencia Vital. Requiere atención inmediata.
- C2: Urgencia / Alta Complejidad: Espera de hasta 30 minutos.
- C3: Mediana complejidad: Espera de hasta 1 hora y 30 minutos.
- C4: Baja complejidad: Espera de hasta 3 horas.
- C5: Atención general: Sin tiempo de espera estimado, depende de la demanda de atención.

Para lograr la correcta gestión de los pacientes se crearon 6 clases:

- Paciente : Esta clase representa a cada uno de los pacientes de manera individual, aquí se fijan los atributos del paciente, siendo estos:
  - Nombre: Nombre del paciente.
  - Apellido : Apellido del paciente.
  - id: ID único de cada paciente.
  - Categoría: Complejidad de atención requerida por el paciente(C1 a C5).
  - Tiempo de llegada: La hora en la cual el paciente es ingresado.
  - Estado: Estado del paciente, si se encuentra en espera, en atención o si ya fue atendido.
  - Área: Área en la cual el paciente será ingresado.
  - Historial de cambios: Aquí se registran las modificaciones de cada paciente, como el cambio de categoría o el cambio de estado del paciente.

---

```

3  public class Paciente implements Comparable<Paciente> {
4      String nombre;
5      String apellido;
6      String id;
7      int categoria; // 1 a 5 (C1 a C5)
8      long tiempoLlegada;
9      String estado; // en_espera, en_atencion, atendido
10     String area; // SAPU, urgencia_adulto, infantil
11     Stack<String> historialCambios = new Stack<>();

```

Figura 1: Clase Paciente

Los metodos utilizados en esta clase son los siguientes:

- long tiempoEsperaActual(long actual): Este método calcula cuantos minutos ha esperado el paciente desde que llegó.

```

23     public long tiempoEsperaActual(long actual) {
24         return (actual - tiempoLlegada) / 60;
25     }

```

Figura 2: Método tiempoEsperaActual

- void registrarCambio(string descripcion): Este método guarda los cambios en el historial, ya sea un cambio de estado del paciente o una re asignación de categoría.

```

27     public void registrarCambio(String descripcion) {
28         historialCambios.push(descripcion);
29     }

```

Figura 3: Método registrarCambio

- String obtenerUltimoCambio(): Este método devuelve el ultimo cambio registrado de cada paciente.

---

```

31     public String obtenerUltimoCambio() {
32         return historialCambios.isEmpty() ? null : historialCambios.pop();
33     }
34

```

Figura 4: Método obtenerUltimoCambio

- `int compareTo(paciente otro)`: Este método compara dos pacientes para ordenarlos en la cola de atención, primero por categoría (mientras menor sea la categoría mayor urgencia de atención) y luego por tiempo de llegada (a mayor tiempo de llegada, mayor prioridad de atención).

```

35     @Override
36     public int compareTo(Paciente otro) {
37         if (this.categoria != otro.categoria) {
38             return Integer.compare(this.categoria, otro.categoria);
39         }
40         return Long.compare(this.tiempoLlegada, otro.tiempoLlegada);
41     }

```

Figura 5: Método compareTo

- **AreaAtencion** : En esta clase se crean las áreas de atención y las cualidades de estas:
  - **Nombre**: Nombre del área de atención.
  - **pacientesHeap**: Una cola de prioridad de pacientes.
  - **capacidad máxima**: Es la capacidad máxima de pacientes que puede tener el área.

```

5     public class AreaAtencion {
6         String nombre;
7         PriorityQueue<Paciente> pacientesHeap;
8         int capacidadMaxima;

```

Figura 6: Clase AreaAtencion

Los metodos que se utilizan en esta clase son los siguientes:

- `AreaAtencion(string nombre, int capacidadMaxima)`; Este método es el constructor de la clase que inicializa el nombre del área, la capacidad máxima de esta y la cola de prioridad.

---

```
10     public AreaAtencion(String nombre, int capacidadMaxima) {
11         this.nombre = nombre;
12         this.capacidadMaxima = capacidadMaxima;
13         this.pacientesHeap = new PriorityQueue<>();
14     }
```

Figura 7: Método AreaAtencion

- void ingresaPaciente(Paciente p): Este método agrega un paciente a la cola de prioridad, si el área de atención no se encuentra llena,

```
16     public void ingresarPaciente(Paciente p) {
17         if (!estaSaturada()) {
18             pacientesHeap.add(p);
19         }
20     }
```

Figura 8: Método ingresaPaciente

- Paciente atenderPaciente(): Este método “atiende” al siguiente paciente en la cola de prioridad el área, para atenderlo, lo elimina de la cola.

```
22     public Paciente atenderPaciente() {
23         return pacientesHeap.poll();
24     }
```

Figura 9: Método atenderPaciente

- bool estaSaturada(): Este método sirve para verificar si el área alcanzo su capacidad máxima de pacientes, retorna “true” si el área ya alcanzo su capacidad máxima, en caso contrario retorna “false”.

```

26     public boolean estaSaturada() {
27         return pacientesHeap.size() >= capacidadMaxima;
28     }

```

Figura 10: Método estaSaturada

- List<Paciente> obtenerPacientesPorHeapSort(): Este método devuelve una lista ordenada de pacientes en el área usando el método "compareTo" de la clase Paciente.

```

30     public List<Paciente> obtenerPacientesPorHeapSort() {
31         List<Paciente> copia = new ArrayList<>(pacientesHeap);
32         copia.sort(c:null); // usa compareTo de Paciente
33         return copia;
34     }

```

Figura 11: Método obtenerPacientesPorHeapSort

- Hospital : En esta clase es donde ocurre todo el flujo de pacientes, esta cuenta con los siguientes atributos:
  - pacientesTotales: Un mapa con todos los pacientes registrados.
  - colaAtencion: Es la cola de prioridad global según la cual se decide a qué paciente atender.
  - areasAtencion: Aquí se asigna a los pacientes por área.

```

7     public class Hospital {
8         Map<String, Paciente> pacientesTotales = new HashMap<>();
9         PriorityQueue<Paciente> colaAtencion = new PriorityQueue<>();
10        Map<String, AreaAtencion> areasAtencion = new HashMap<>();
11        List<Paciente> pacientesAtendidos = new ArrayList<>();

```

Figura 12: Clase Hospital

Los métodos utilizados en esta clase son los siguientes:

- Hospital(): Es el constructor de la clase, inicializa las áreas de atención.

---

```

13 public Hospital() {
14     areasAtencion.put(key:"SAPU", new AreaAtencion(nombre:"SAPU", capacidadMaxima:100));
15     areasAtencion.put(key:"urgencia_adulto", new AreaAtencion(nombre:"urgencia_adulto", capacidadMaxima:100));
16     areasAtencion.put(key:"infantil", new AreaAtencion(nombre:"infantil", capacidadMaxima:100));
17 }

```

Figura 13: Método Hospital

- void registrarPaciente(Paciente P): Este método registra un paciente, lo agrega al método "pacientesTotales" por su id, y también lo agrega al método "colaAtencion" según la prioridad del paciente.

```

19 public void registrarPaciente(Paciente p) {
20     pacientesTotales.put(p.id, p);
21     colaAtencion.add(p);
22 }

```

Figura 14: Método registrarPaciente

- void reasignarCategoria(String id, int nuevaCategoria): Este método cambia la categoría de un paciente, lo saca de la cola, cambia la categoría, registra el cambio y lo vuelve a agregar.

```

24 public void reasignarCategoria(String id, int nuevaCategoria) {
25     Paciente p = pacientesTotales.get(id);
26     if (p != null) {
27         colaAtencion.remove(p);
28         p.registrarCambio("Reasignado de C" + p.categoria + " a C" + nuevaCategoria);
29         p.categoria = nuevaCategoria;
30         colaAtencion.add(p);
31     }
32 }

```

Figura 15: Método reasignarCategoria

- Paciente atenderSiguiente(long actual): Atiende al siguiente paciente de la cola, primero lo saca de la cola, lo marca como atendido, lo agrega a la lista "pacientesAtendidos" y lo asigna a su área respectiva si es que esta tiene espacio.



---

```

34     public Paciente atenderSiguiente(long actual) {
35         Paciente p = colaAtencion.poll();
36         if (p != null) {
37             p.estado = "atendido";
38             pacientesAtendidos.add(p);
39             areasAtencion.get(p.area).ingresarPaciente(p);
40         }
41         return p;
42     }

```

Figura 16: Método atenderSiguiente

- List<Paciente> obtenerPacientesPorCategoria(int categoria): Este método busca y devuelve pacientes en espera de una categoría específica.

```

44     public List<Paciente> obtenerPacientesPorCategoria(int categoria) {
45         List<Paciente> resultado = new ArrayList<>();
46         for (Paciente p : colaAtencion) {
47             if (p.categoria == categoria)
48                 resultado.add(p);
49         }
50         return resultado;
51     }

```

Figura 17: Método obtenerPacientesPorCategoria

- AreaAtencion obtenerArea(string nombre): Este método retorna el objeto "AreaAtencion" dado su nombre.

```

53     public AreaAtencion obtenerArea(String nombre) {
54         return areasAtencion.get(nombre);
55     }

```

Figura 18: Método obtenerArea

- GeneradorPacientes : Esta clase es la encargada de generar pacientes aleatorios, contiene dos listas fijas, una de nombres y una de apellidos para generar nombres aleatorios de pacientes "static String[] nombres" y "static String[] apellidos" respectivamente. El método que contiene esta clase es el siguiente:

---

```

5  public class GeneradorPacientes {
6      static String[] nombres = { "Ana", "Luis", "Carlos", "Marta", "Pedro" };
7      static String[] apellidos = { "Pérez", "Gómez", "Soto", "Díaz", "Morales" };

```

Figura 19: Clase GeneradorPacientes

- static List<Paciente> generarPacientes(int N, long timestampBase): El método genera "N" pacientes aleatorios y les asigna:
  - Nombre y Apellido.
  - Id único.
  - Categoría según distribución realista de manera porcentual()
  - Área de manera ciclica alternando entre: "SAPU", "urgencia\_adulto" y "infantil"
- SimuladorUrgencia : Esta clase realiza una simulación de 24 horas (1440 minutos), los métodos utilizados para realizar la simulacion son los siguientes:
  - SimuladorUrgencia(Hospital hospital, List<Paciente> pacientes): Este método es el constructor de la clase, que recibe el hospital y la lista de pacientes que seran utilizados para la simulación.

```

6  public class SimuladorUrgencia {
7      Hospital hospital;
8      List<Paciente> pacientes;
9      Map<Integer, List<Long>> tiemposPorCategoria = new HashMap<>();
10     List<Paciente> fueraDeTiempo = new ArrayList<>();

```

Figura 20: Método SimuladorUrgencia

- void simular(int pacientesPorDia): Este método ejecuta la simulación de un día Completo(1440 minutos):
  - Cada 10 minutos registra un paciente.
  - Cada 15 minutos atiende un paciente.
  - Cada 3 pacientes registrados, atiende 2 pacientes adicionales.

Al final el método calcula el tiempo de espera real para cada paciente y si fue atendido dentro del límite permitido por su categoría.

```

17 public void simular(int pacientesPorDia) {
18     long tiempo = 0;
19     int ingresados = 0;
20
21     for (int minuto = 0; minuto < 1440; minuto++) {
22         tiempo = minuto * 60;
23
24         if (minuto % 10 == 0 && ingresados < pacientes.size()) {
25             Paciente nuevo = pacientes.get(ingresados);
26             hospital.registrarPaciente(nuevo);
27             ingresados++;
28
29             if (ingresados % 3 == 0) {
30                 hospital.atenderSiguiente(tiempo);
31                 hospital.atenderSiguiente(tiempo);
32             }
33         }
34
35         if (minuto % 15 == 0) {
36             hospital.atenderSiguiente(tiempo);
37         }
38     }
39
40     for (Paciente p : hospital.pacientesAtendidos) {
41         long espera = (p.tiempoEsperaActual(tiempo));
42         tiemposPorCategoria.computeIfAbsent(p.categoria, k -> new ArrayList<>()).add(espera);
43
44         long limite = switch (p.categoria) {
45             case 1 -> 0;
46             case 2 -> 30;
47             case 3 -> 90;
48             case 4 -> 180;
49             default -> Long.MAX_VALUE;
50         };
51         if (espera > limite)
52             fueraDeTiempo.add(p);
53     }
54 }

```

Figura 21: Método simular

- void imprimirEstadisticas(): Este método imprime el tiempo promedio de espera por categoría y la cantidad de pacientes que esperaron más de lo permitido.

```

56 public void imprimirEstadisticas() {
57     for (int cat = 1; cat <= 5; cat++) {
58         List<Long> tiempos = tiemposPorCategoria.getDefault(cat, new ArrayList<>());
59         double promedio = tiempos.stream().mapToLong(Long::longValue).average().orElse(0);
60         System.out.println("C" + cat + " promedio espera: " + promedio + " minutos");
61     }
62     System.out.println("Pacientes fuera de tiempo: " + fueraDeTiempo.size());
63 }

```

Figura 22: Método imprimirEstadisticas

- 
- Main: Esta clase genera 144 pacientes(1 cada 10 minutos por 1440 minutos o 24 horas), luego ejecuta la simulación completa y muestra las estadísticas finales.

```
3  public class Main {  
    Run | Debug  
4      public static void main(String[] args) {  
5          long base = 0L;  
6          int N = 144;  
7  
8          Hospital hospital = new Hospital();  
9          List<Paciente> pacientes = GeneradorPacientes.generarPacientes(N, base);  
10         SimuladorUrgencia simulador = new SimuladorUrgencia(hospital, pacientes);  
11         simulador.simular(N);  
12         simulador.imprimirEstadisticas();  
13     }  
14 }
```

Figura 23: Clase Main

## 3. Experimentación y Resultados

La experimentación consistió en la simulación del funcionamiento de una sala de urgencias hospitalaria durante un período de 24 horas, modelando tanto la llegada aleatoria de pacientes como su atención según prioridad médica.

### 3.1. Generación de Pacientes

Se generaron 144 pacientes, uno cada 10 minutos, simulando un día completo. Los pacientes fueron creados con la clase `GeneradorPacientes`, que asigna:

- Nombre y apellido aleatorio a partir de listas fijas.
- ID único incremental.
- Categoría de urgencia basada en la siguiente distribución:
  - C1: 10
  - C2: 15
  - C3: 18
  - C4: 27
  - C5: 30
- Área de atención rotativa entre SAPU, urgencia\_adulto e infantil.
- Hora de llegada cada 600 segundos (10 minutos).
- Estado inicial en\_espera.

---

## 3.2. Configuración de la Simulación

La clase SimuladorUrgencia gestiona la simulación minuto a minuto durante un total de 1440 minutos. Las reglas principales implementadas fueron:

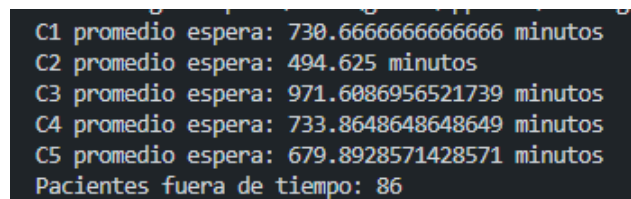
- Ingreso: cada 10 minutos entra un nuevo paciente.
- Atención periódica: cada 15 minutos se atiende un paciente.
- Atención por acumulación: cada 3 ingresos se atienden 2 pacientes adicionales.

Cada paciente atendido tiene su estado actualizado a atendido y es transferido al área correspondiente. El sistema registra los tiempos de espera, pacientes excedidos y categorías atendidas.

1. Seguimiento individual: se monitorearon pacientes de categoría C4 para verificar si eran atendidos dentro del límite de 180 minutos.
2. Promedio por categoría: el sistema calculó el tiempo promedio de espera por cada categoría (C1 a C5), usando listas agrupadas por categoría.
3. Pacientes fuera de tiempo: se identificaron aquellos pacientes que excedieron el tiempo máximo de espera permitido según su categoría.
4. Carga del sistema: se observó cómo se comporta la cola de atención con el ingreso regular de pacientes y su distribución en áreas.

## 3.3. Resultados

Estos resultados se obtuvieron al correr la simulación una vez:



```
C1 promedio espera: 730.6666666666666 minutos  
C2 promedio espera: 494.625 minutos  
C3 promedio espera: 971.6086956521739 minutos  
C4 promedio espera: 733.8648648648649 minutos  
C5 promedio espera: 679.8928571428571 minutos  
Pacientes fuera de tiempo: 86
```

Figura 24: Resultados

---

## 4. Conclusiones

En este laboratorio se aplicaron como las colas de prioridad, mapas, montones(heaps) y pilas, en un contexto realista y aplicable al mundo real. Este laboratorio ayudó a comprender la importancia de una correcta elección de estructuras adecuadas para cada caso, ya que estas influyen directamente en el rendimiento del algoritmo creado. Por otro lado el trabajar con tiempos incita a desarrollar algoritmos con un enfoque orientado a la eficiencia.

<https://github.com/gabovrs/lab4eda>