

# Ejercicios resueltos de Programación de Servicios y Procesos

Los 13 ejercicios de Programación de Servicios y Procesos de la web, tema a tema: cada uno con su enunciado, los datos que necesitas (código de partida, ejemplos de entrada y salida o la base de datos) y, al final, la solución explicada.

## Índice

### Programación Concurrente

#### 1. Coordinador de procesos

#### 2. Planificador de CPU: FCFS, SJF y Round Robin

### Hilos y Multithreading en Java

#### 3. Suma en paralelo con hilos

#### 4. Productor-consumidor con un búfer acotado

### Sockets en Java

#### 5. Servidor de un protocolo de texto

#### 6. Mensajes fiables sobre UDP

### Desarrollo de Servicios en Red

#### 7. Una API REST con HttpServer

#### 8. Disyuntor (circuit breaker) para un servicio inestable

### Programación Segura en Java

#### 9. Guardar contraseñas con PBKDF2

#### 10. Tokens firmados con HMAC (como un JWT)

### Ejercicios largos

#### 11. Contador de palabras en paralelo con un pool de hilos

#### 12. Servidor de órdenes con sockets e hilos

#### 13. Almacén de contraseñas seguro con PBKDF2 y sal

# Programación Concurrente

## 1. Coordinador de procesos

Difícil · Programación Concurrente · [apuntesdam.com/subject/prog-servicios/topic/concurrencia](https://apuntesdam.com/subject/prog-servicios/topic/concurrencia)

El programa puede funcionar en dos modos. Si recibe los argumentos «trabajador desde hasta», es un proceso hijo: cuenta los primos de ese tramo y escribe el número (ya está hecho). Si no, es el coordinador: lee un límite y un número de procesos, reparte el intervalo [1, límite] en tantos tramos iguales como procesos (el último se queda con el resto) y lanza un proceso hijo por tramo con ProcessBuilder, ejecutando esta misma clase. Primero lanza todos y después lee la salida de cada uno en orden y espera a que termine. Muestra «Proceso i [desde, hasta]: n primos» y, al final, «Total: N primos hasta límite».

**Código de partida** (java)

```
import java.io.*;
import java.util.*;

public class Main {
    static boolean esPrimo(long n) {
        if (n < 2) return false;
        for (long d = 2; d * d <= n; d++) if (n % d == 0) return false;
        return true;
    }

    static long contar(long desde, long hasta) {
        long c = 0;
        for (long i = desde; i <= hasta; i++) if (esPrimo(i)) c++;
        return c;
    }

    public static void main(String[] args) throws Exception {
        // Modo trabajador: el proceso hijo cuenta su tramo y escribe el resultado
        if (args.length == 3 && args[0].equals("trabajador")) {
            System.out.println(contar(Long.parseLong(args[1]), Long.parseLong(args[2])));
            return;
        }
        Scanner sc = new Scanner(System.in);
        long limite = sc.nextLong();
        int procesos = sc.nextInt();
        // TODO: repartir [1, limite] en tramos, lanzar un proceso hijo por tramo,
        //         leer su resultado y mostrar el resumen
    }
}
```

### Ejemplo: 10 000 con 4 procesos

ENTRADA

10000 4

SALIDA ESPERADA

Proceso 1 [1, 2500]: 367 primos  
Proceso 2 [2501, 5000]: 302 primos  
Proceso 3 [5001, 7500]: 281 primos  
Proceso 4 [7501, 10000]: 279 primos  
Total: 1229 primos hasta 10000

### Ejemplo: Reparto con resto

ENTRADA

100 3

SALIDA ESPERADA

Proceso 1 [1, 33]: 11 primos  
Proceso 2 [34, 66]: 7 primos  
Proceso 3 [67, 100]: 7 primos  
Total: 25 primos hasta 100

## 2. Planificador de CPU: FCFS, SJF y Round Robin

Muy difícil · Programación Concurrente · [apuntesdam.com/subject/prog-servicios/topic/concurrencia](https://apuntesdam.com/subject/prog-servicios/topic/concurrencia)

El sistema operativo decide qué proceso usa la CPU en cada momento. La entrada es el quantum y, después, cada proceso con su identificador (una letra), su instante de llegada y su ráfaga de CPU. Implementa los tres algoritmos devolviendo la línea de tiempo: un carácter por unidad de tiempo con la letra del proceso que se ejecuta, o «-» si la CPU está libre. FCFS ejecuta los procesos enteros por orden de llegada (a igualdad, en el orden de la entrada). SJF, no expropiativo, cada vez que la CPU queda libre elige, de los que ya han llegado, el de ráfaga más corta (a igualdad, el primero de la entrada). Round Robin da turnos de como mucho quantum unidades con una cola FIFO: los procesos que llegan durante un turno entran en la cola antes que el proceso que acaba de ser expulsado. El método informe, ya hecho, calcula el tiempo de espera de cada proceso (fin – llegada – ráfaga) y la media.

**Código de partida** (java)

```
import java.util.*;

public class Main {
    record Proceso(String id, int llegada, int rafaga) { }

    // Devuelve la línea de tiempo: una letra (id) por unidad de tiempo, o '-' si la CPU está libre
    static String fcfs(List<Proceso> ps) {
        // TODO: el primero que llega se ejecuta entero
        return "";
    }

    static String sjf(List<Proceso> ps) {
        // TODO: no expropiativo; de los que han llegado, el de ráfaga más corta
        return "";
    }

    static String roundRobin(List<Proceso> ps, int quantum) {
        // TODO: turnos de como mucho quantum unidades
        return "";
    }

    // Calcula y muestra los tiempos de espera a partir de la línea de tiempo
    static void informe(String nombre, String linea, List<Proceso> ps) {
        StringBuilder sb = new StringBuilder(nombre + ": " + linea + " |");
        double suma = 0;
        for (Proceso p : ps) {
            int fin = linea.lastIndexOf(p.id().charAt(0)) + 1;
            int espera = fin - p.llegada() - p.rafaga();
            suma += espera;
            sb.append(" ").append(p.id()).append("=").append(espera);
        }
        sb.append(String.format(Locale.US, " | media %.2f", suma / ps.size()));
        System.out.println(sb);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int quantum = sc.nextInt();
        List<Proceso> ps = new ArrayList<>();
        while (sc.hasNext()) ps.add(new Proceso(sc.next(), sc.nextInt(), sc.nextInt()));
        informe("FCFS", fcfs(ps), ps);
        informe("SJF ", sjf(ps), ps);
        informe("RR ", roundRobin(ps, quantum), ps);
    }
}
```

### Ejemplo: Ejemplo de clase

#### ENTRADA

```
2
A 0 5
B 1 3
C 2 1
D 3 2
```

#### SALIDA ESPERADA

|       |              |                 |            |
|-------|--------------|-----------------|------------|
| FCFS: | AAAAABBCDD   | A=0 B=4 C=6 D=6 | media 4.00 |
| SJF : | AAAAACDDBBB  | A=0 B=7 C=3 D=3 | media 3.25 |
| RR :  | AABBCAADDDBA | A=6 B=6 C=2 D=4 | media 4.50 |

### Ejemplo: CPU libre al principio

#### ENTRADA

```
3
A 2 4
B 3 2
C 10 1
```

#### SALIDA ESPERADA

|       |             |             |            |
|-------|-------------|-------------|------------|
| FCFS: | --AAAABB--C | A=0 B=3 C=0 | media 1.00 |
| SJF : | --AAAABB--C | A=0 B=3 C=0 | media 1.00 |
| RR :  | --AAABBA--C | A=2 B=2 C=0 | media 1.33 |

## Hilos y Multithreading en Java

### 3. Suma en paralelo con hilos

Difícil · Hilos y Multithreading en Java · [apuntesdam.com/subject/prog-servicios/topic/hilos](https://apuntesdam.com/subject/prog-servicios/topic/hilos)

La primera cifra de la entrada es el número de hilos y el resto, los números que hay que sumar. Reparte el array en tantos bloques consecutivos como hilos, lo más iguales posible: si no es divisible, los primeros bloques tienen un elemento más (con 10 números y 3 hilos: 4, 3 y 3). Cada hilo suma su bloque y guarda el resultado en su posición de parciales. El main debe esperar a que terminen todos antes de mostrar los resultados (esa parte ya está hecha). Un hilo puede quedarse sin elementos: entonces su parcial es 0.

#### Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);
        int hilos = sc.nextInt();
        List<Long> lista = new ArrayList<>();
        while (sc.hasNextLong()) lista.add(sc.nextLong());
        long[] datos = lista.stream().mapToLong(Long::longValue).toArray();

        long[] parciales = new long[hilos]; // cada hilo escribe solo en su posición
        // TODO: crear los hilos, repartir el array en bloques, arrancarlos y esperar a que terminen

        long total = 0;
        for (int i = 0; i < hilos; i++) {
            System.out.println("Hilo " + i + ": " + parciales[i]);
            total += parciales[i];
        }
        System.out.println("Total: " + total);
    }
}
```

### Ejemplo: 3 hilos, 10 números

#### ENTRADA

```
3
1 2 3 4 5 6 7 8 9 10
```

#### SALIDA ESPERADA

```
Hilo 0: 10
Hilo 1: 18
Hilo 2: 27
Total: 55
```

### Ejemplo: Más hilos que números

#### ENTRADA

```
4
100 200 300
```

#### SALIDA ESPERADA

```
Hilo 0: 100
Hilo 1: 200
Hilo 2: 300
Hilo 3: 0
Total: 600
```

## 4. Productor-consumidor con un búfer acotado

Muy difícil · Hilos y Multithreading en Java · [apuntesdam.com/subject/prog-servicios/topic/hilos](https://apuntesdam.com/subject/prog-servicios/topic/hilos)

Varios productores dejan números en un búfer de capacidad limitada y un consumidor los va sacando. Implementa los métodos sincronizados de `BufferAcotado` con `wait` y `notifyAll`: poner espera mientras el búfer está lleno y sacar espera mientras está vacío; los dos avisan a los demás hilos cuando cambian el estado. Usa un array circular (inicio y cantidad) para que sacar sea inmediato. El main, ya hecho, lanza los productores, espera a que terminen, mete un FIN para el consumidor y muestra cuántos elementos de cada productor se consumieron y la suma: si no se pierde ni se repite ningún elemento, cada productor aporta todos los suyos.

## Código de partida (java)

```
import java.util.*;

class BufferAcotado {
    private final int[] datos;
    private int inicio = 0, cantidad = 0;

    BufferAcotado(int capacidad) {
        datos = new int[capacidad];
    }

    // Espera mientras el búfer está lleno
    public synchronized void poner(int v) throws InterruptedException {
        // TODO
    }

    // Espera mientras el búfer está vacío
    public synchronized int sacar() throws InterruptedException {
        // TODO
        return -1;
    }
}

public class Main {
    static final int FIN = -1; // «píldora venenosa»: avisa al consumidor de que no
                               hay más

    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);
        int capacidad = sc.nextInt(), productores = sc.nextInt(), porProductor = sc.nextInt();
        BufferAcotado buffer = new BufferAcotado(capacidad);
        int[] consumidosDe = new int[productores];
        long[] suma = new long[1];

        List<Thread> hilos = new ArrayList<>();
        for (int p = 0; p < productores; p++) {
            final int id = p;
            hilos.add(new Thread(() -> {
                try {
                    // Cada valor codifica su productor: id * 1000 + número
                    for (int k = 1; k <= porProductor; k++) buffer.poner(id * 1000 + k);
                } catch (InterruptedException e) { Thread.currentThread().interrupt(); }
            }));
        }
        Thread consumidor = new Thread(() -> {
            try {
                int v;
                while ((v = buffer.sacar()) != FIN) {
                    consumidosDe[v / 1000]++;
                    suma[0] += v % 1000;
                }
            } catch (InterruptedException e) { Thread.currentThread().interrupt(); }
        });
        consumidor.start();
        for (Thread h : hilos) h.start();
        for (Thread h : hilos) h.join();
        buffer.poner(FIN);
        consumidor.join();

        for (int p = 0; p < productores; p++) System.out.println("Productor " + p + ": " +
            consumidosDe[p] + " elementos consumidos");
        System.out.println("Suma de los números: " + suma[0]);
    }
}
```

### Ejemplo: Capacidad 2, 3 productores

#### ENTRADA

2 3 50

#### SALIDA ESPERADA

Productor 0: 50 elementos consumidos  
Productor 1: 50 elementos consumidos  
Productor 2: 50 elementos consumidos  
Suma de los números: 3825

### Ejemplo: Capacidad 1

#### ENTRADA

1 4 25

#### SALIDA ESPERADA

Productor 0: 25 elementos consumidos  
Productor 1: 25 elementos consumidos  
Productor 2: 25 elementos consumidos  
Productor 3: 25 elementos consumidos  
Suma de los números: 1300

## Sockets en Java

### 5. Servidor de un protocolo de texto

Difícil · Sockets en Java · [apuntesdam.com/subject/prog-servicios/topic/sockets](https://apuntesdam.com/subject/prog-servicios/topic/sockets)

El main ya arranca un servidor TCP en un puerto libre y un cliente que le envía cada línea de la entrada y muestra la respuesta. Implementa atender, que gestiona la conexión de un cliente siguiendo este protocolo (una orden por línea, una respuesta por línea): «SUMA n1 n2 ...» responde «OK total» (o «ERROR números no válidos»); «MAYUS texto» responde «OK TEXTO»; «INVERTIR texto» responde el texto al revés; «CONTAR» responde «OK n», con el número de órdenes atendidas antes en esta conexión (todas, incluidas las erróneas y otros CONTAR); «ADIOS» responde «OK hasta luego» y cierra. Las órdenes no distinguen mayúsculas y cualquier otra da «ERROR orden desconocida: X». Usa UTF-8 en los dos sentidos.

## Código de partida (java)

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Main {
    // Atiende a UN cliente: lee órdenes línea a línea y responde a cada una
    static void atender(Socket cliente) throws IOException {
        // TODO: leer cada línea, responder según el protocolo y terminar con ADIOS
    }

    public static void main(String[] args) throws Exception {
        try (ServerSocket servidor = new ServerSocket(0)) {           // puerto libre cualquiera
            Thread hiloServidor = new Thread(() -> {
                try (Socket c = servidor.accept()) {
                    atender(c);
                } catch (IOException e) {
                    System.out.println("Error del servidor: " + e.getMessage());
                }
            });
            hiloServidor.start();

            // Cliente: envía cada línea de la entrada y muestra la respuesta
            Scanner sc = new Scanner(System.in);
            try (Socket s = new Socket("localhost", servidor.getLocalPort());
                PrintWriter out = new PrintWriter(new OutputStreamWriter(s.getOutputStream(), "UTF-8"),
                    true);
                BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream(), "UTF-8"))) {
                while (sc.hasNextLine()) {
                    String orden = sc.nextLine();
                    out.println(orden);
                    String r;
                    try {
                        r = in.readLine();
                    } catch (IOException e) {
                        r = null;                                     // el servidor cerró la conexión
                    }
                    System.out.println("> " + orden);
                    System.out.println("< " + (r == null ? "(conexión cerrada)" : r));
                    if (r == null || orden.equals("ADIOS")) break;
                }
            }
            hiloServidor.join();
        }
    }
}
```

## Ejemplo: Órdenes básicas

### ENTRADA

```
SUMA 2 3 4
MAYUS hola mundo
INVERTIR socket
CONTAR
ADIOS
```

### SALIDA ESPERADA

```
> SUMA 2 3 4
< OK 9
> MAYUS hola mundo
< OK HOLA MUNDO
> INVERTIR socket
< OK tekcos
> CONTAR
< OK 3
> ADIOS
< OK hasta luego
```

### Ejemplo: Errores

#### ENTRADA

```
suma 1 dos  
BORRAR todo  
CONTAR  
adios
```

#### SALIDA ESPERADA

```
> suma 1 dos  
< ERROR números no válidos  
> BORRAR todo  
< ERROR orden desconocida: BORRAR  
> CONTAR  
< OK 2  
> adios  
< OK hasta luego
```

## 6. Mensajes fiables sobre UDP

Muy difícil · Sockets en Java · [apuntesdam.com/subject/prog-servicios/topic/sockets](https://apuntesdam.com/subject/prog-servicios/topic/sockets)

UDP no garantiza que los paquetes lleguen. El servidor de este reto lo simula: pierde uno de cada cuatro paquetes que recibe y una de cada tres respuestas. Implementa `enviarFiable` con el protocolo de parada y espera (stop-and-wait) que usan TCP y muchos juegos: envía el datagrama «seq:texto», espera como mucho `ESPERA_MS` milisegundos un datagrama «ACK seq» y, si no llega, reenvía el mismo mensaje, hasta `MAX_INTENTOS` veces. Devuelve el número de intentos que han hecho falta, o `-1`. Ignora cualquier ACK de otro número de secuencia. El servidor descarta los mensajes duplicados (cuando se pierde un ACK, el cliente reenvía algo que ya había llegado) gracias al número de secuencia, así que al final debe haber recibido todos los mensajes, en orden y una sola vez.

## Código de partida (java)

```
import java.io.*;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.util.*;

public class Main {
    static final int ESPERA_MS = 150, MAX_INTENTOS = 5;

    // ---- Servidor (ya hecho): pierde paquetes a propósito para simular una red mala
    static class Servidor extends Thread {
        final DatagramSocket ds;
        final List<String> recibidos = new ArrayList<>();
        int paquetes = 0, duplicados = 0, ultimoSeq = 0;

        Servidor() throws SocketException { ds = new DatagramSocket(0); }

        public void run() {
            byte[] buf = new byte[1024];
            try {
                while (true) {
                    DatagramPacket p = new DatagramPacket(buf, buf.length);
                    ds.receive(p);
                    paquetes++;
                    if (paquetes % 4 == 0) continue; // se pierde el paquete que llega
                    String msg = new String(p.getData(), 0, p.getLength(), StandardCharsets.UTF_8);
                    int seq = Integer.parseInt(msg.substring(0, msg.indexOf(':')));
                    String datos = msg.substring(msg.indexOf(':') + 1);
                    if (seq == ultimoSeq) duplicados++; // ya lo teníamos: era una retransmisión
                    else if (seq == ultimoSeq + 1) { ultimoSeq = seq; if (!datos.equals("FIN"))
                        recibidos.add(datos); }
                    if (paquetes % 3 == 0) continue; // se pierde el ACK de respuesta
                    byte[] ack = ("ACK " + seq).getBytes(StandardCharsets.UTF_8);
                    ds.send(new DatagramPacket(ack, ack.length, p.getSocketAddress()));
                    if (datos.equals("FIN")) return;
                }
            } catch (IOException e) { }
        }
    }

    // Envía un mensaje y espera su ACK; si no llega a tiempo, lo reenvía.
    // Devuelve el número de intentos, o -1 si se agotan.
    static int enviarFiable(DatagramSocket ds, SocketAddress destino, int seq, String texto) throws
        IOException {
        // TODO
        byte[] datos = (seq + ":" + texto).getBytes(StandardCharsets.UTF_8);
        ds.send(new DatagramPacket(datos, datos.length, destino));
        return 1;
    }

    public static void main(String[] args) throws Exception {
        Servidor srv = new Servidor();
        srv.start();
        SocketAddress destino = new InetSocketAddress("localhost", srv.ds.getLocalPort());
        Scanner sc = new Scanner(System.in);
        List<String> mensajes = new ArrayList<>();
        while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) mensajes.add(l); }
        mensajes.add("FIN");
        try (DatagramSocket ds = new DatagramSocket()) {
            for (int i = 0; i < mensajes.size(); i++) {
                int intentos = enviarFiable(ds, destino, i + 1, mensajes.get(i));
                System.out.println("Mensaje " + (i + 1) + " «" + mensajes.get(i) + "»: "
                    + (intentos < 0 ? "sin respuesta" : intentos + (intentos == 1 ? " intento" : " intentos")));
            }
        }
        srv.join(2000);
        srv.ds.close();
        System.out.println("El servidor recibió en orden: " + String.join(" | ", srv.recibidos));
        System.out.println("Retransmisiones duplicadas descartadas: " + srv.duplicados);
    }
}
```

```
}  
}
```

### Ejemplo: Cuatro mensajes

#### ENTRADA

```
hola  
que  
tal  
estás
```

#### SALIDA ESPERADA

```
Mensaje 1 «hola»: 1 intento  
Mensaje 2 «que»: 1 intento  
Mensaje 3 «tal»: 3 intentos  
Mensaje 4 «estás»: 2 intentos  
Mensaje 5 «FIN»: 3 intentos  
El servidor recibió en orden: hola | que | tal |  
estás  
Retransmisiones duplicadas descartadas: 3
```

### Ejemplo: Uno solo

#### ENTRADA

```
ping
```

#### SALIDA ESPERADA

```
Mensaje 1 «ping»: 1 intento  
Mensaje 2 «FIN»: 1 intento  
El servidor recibió en orden: ping  
Retransmisiones duplicadas descartadas: 0
```

## Desarrollo de Servicios en Red

### 7. Una API REST con HttpServer

Difícil · Desarrollo de Servicios en Red · [apuntesdam.com/subject/prog-servicios/topic/servicios-red-java](https://apuntesdam.com/subject/prog-servicios/topic/servicios-red-java)

Completa manejar para que el servidor ofrezca una API REST de tareas. GET /tareas devuelve 200 y la lista en JSON, ordenada por id: [{"id":1,"texto":"..."}]. POST /tareas con el texto como cuerpo crea una tarea con el siguiente id, responde 201 con la tarea en JSON y la cabecera Location; si el texto está vacío, 400 con {"error":"texto vacío"}. GET /tareas/{id} devuelve 200 con la tarea; DELETE /tareas/{id} la borra y responde 204 sin cuerpo. Si el id no es un número: 400 {"error":"id no válido"}; si no existe: 404 {"error":"no existe la tarea N"}. Cualquier otro método: 405 sin cuerpo. Usa los métodos responder y json, ya hechos. El main envía con HttpClient las peticiones de la entrada y muestra el código y el cuerpo.

## Código de partida (java)

```
import com.sun.net.httpserver.*;
import java.io.*;
import java.net.*;
import java.net.http.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicInteger;

public class Main {
    static final Map<Integer, String> tareas = new ConcurrentSkipListMap<>(); // id → texto, ordenadas
    static final AtomicInteger siguienteId = new AtomicInteger(1);

    static void responder(HttpExchange ex, int estado, String cuerpo) throws IOException {
        byte[] b = cuerpo.getBytes(StandardCharsets.UTF_8);
        ex.getResponseHeaders().set("Content-Type", "application/json; charset=utf-8");
        ex.sendResponseHeaders(estado, b.length == 0 ? -1 : b.length);
        if (b.length > 0) ex.getResponseBody().write(b);
        ex.close();
    }

    static String json(int id, String texto) {
        return "{\"id\":\"" + id + "\",\"texto\":\"" + texto.replace("\\", "\\\\"").replace("\"", "\\\"") +
            "\"}\"";
    }

    // Atiende /tareas y /tareas/{id}
    static void manejar(HttpExchange ex) throws IOException {
        // TODO: GET /tareas, POST /tareas, GET /tareas/{id}, DELETE /tareas/{id} y los errores
        responder(ex, 501, "");
    }

    public static void main(String[] args) throws Exception {
        HttpServer srv = HttpServer.create(new InetSocketAddress("localhost", 0), 0);
        srv.createContext("/tareas", Main::manejar);
        srv.start();
        String base = "http://localhost:" + srv.getAddress().getPort();

        HttpClient cliente = HttpClient.newHttpClient();
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split(" ", 3); // MÉTODO ruta [cuerpo]
            HttpRequest.BodyPublisher cuerpo = t.length > 2
                ? HttpRequest.BodyPublishers.ofString(t[2]) : HttpRequest.BodyPublishers.noBody();
            HttpRequest req = HttpRequest.newBuilder(URI.create(base + t[1])).method(t[0],
                cuerpo).build();
            HttpResponse<String> r = cliente.send(req, HttpResponse.BodyHandlers.ofString());
            System.out.println(t[0] + " " + t[1] + " → " + r.statusCode() + (r.body().isEmpty() ? "" : " "
                + r.body()));
        }
        srv.stop(0);
    }
}
```

## Ejemplo: Crear, listar y borrar

### ENTRADA

```
POST /tareas Comprar pan
POST /tareas Estudiar PSP
GET /tareas
DELETE /tareas/1
GET /tareas
```

### SALIDA ESPERADA

```
POST /tareas → 201 {"id":1,"texto":"Comprar pan"}
POST /tareas → 201 {"id":2,"texto":"Estudiar PSP"}
GET /tareas → 200 [{"id":1,"texto":"Comprar pan"}, {"id":2,"texto":"Estudiar PSP"}]
DELETE /tareas/1 → 204
GET /tareas → 200 [{"id":2,"texto":"Estudiar PSP"}]
```

### Ejemplo: Errores

#### ENTRADA

```
GET /tareas/7
GET /tareas/abc
POST /tareas
PUT /tareas
DELETE /tareas/1
```

#### SALIDA ESPERADA

```
GET /tareas/7 → 404 {"error":"no existe la tarea 7"}
GET /tareas/abc → 400 {"error":"id no válido"}
POST /tareas → 400 {"error":"texto vacío"}
PUT /tareas → 405
DELETE /tareas/1 → 404 {"error":"no existe la tarea 1"}
```

## 8. Disyuntor (circuit breaker) para un servicio inestable

Muy difícil · Desarrollo de Servicios en Red · [apuntesdam.com/subject/prog-servicios/topic/servicios-red-java](https://apuntesdam.com/subject/prog-servicios/topic/servicios-red-java)

Cuando un servicio remoto se cae, seguir llamándolo empeora las cosas: cada petición espera un tiempo de espera completo y sobrecarga al servicio cuando intenta recuperarse. El patrón disyuntor lo evita con tres estados. CERRADO: las llamadas pasan; tras umbralFallos fallos seguidos, pasa a ABIERTO y el contador de fallos vuelve a 0. ABIERTO: se rechazan las llamadas sin hacerlas durante espera instantes desde que se abrió. Pasado ese tiempo, la siguiente llamada se permite como prueba y el estado pasa a SEMIABIERTO. SEMIABIERTO: si la prueba sale bien, vuelve a CERRADO (y se reinician los fallos); si falla, vuelve a ABIERTO desde ese instante (aunque sea un solo fallo). Un éxito en CERRADO también reinicia el contador de fallos. Implementa permitir y registrar. La entrada es el umbral, la espera y una cadena con el resultado que daría el servicio en cada instante (E o F); el main simula el tiempo.

## Código de partida (java)

```
import java.util.*;

class CircuitBreaker {
    enum Estado { CERRADO, ABIERTO, SEMIABIERTO }

    private final int umbralFallos;        // fallos seguidos para abrir
    private final int espera;              // instantes que permanece abierto
    private Estado estado = Estado.CERRADO;
    private int fallosSeguidos = 0;
    private int abiertoDesde = 0;

    CircuitBreaker(int umbralFallos, int espera) {
        this.umbralFallos = umbralFallos;
        this.espera = espera;
    }

    Estado estado() { return estado; }

    // ¿Se puede llamar al servicio en el instante t? (puede pasar de ABIERTO a SEMIABIERTO)
    boolean permitir(int t) {
        // TODO
        return true;
    }

    // Se informa del resultado de una llamada que se permitió
    void registrar(boolean exito, int t) {
        // TODO
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircuitBreaker cb = new CircuitBreaker(sc.nextInt(), sc.nextInt());
        String resultados = sc.next();          // una letra por instante: E = el servicio responde,
        F = falla
        int llamadas = 0, rechazadas = 0;
        for (int t = 0; t < resultados.length(); t++) {
            String linea;
            if (cb.permidir(t)) {
                llamadas++;
                boolean ok = resultados.charAt(t) == 'E';
                cb.registrar(ok, t);
                linea = (ok ? "llamada OK " : "llamada FALLA");
            } else {
                rechazadas++;
                linea = "rechazada ";
            }
            System.out.println("t=" + t + " " + linea + " → " + cb.estado());
        }
        System.out.println("Llamadas al servicio: " + llamadas + ", rechazadas sin llamar: " +
            rechazadas);
    }
}
```

### Ejemplo: Caída y recuperación

#### ENTRADA

```
3 4
EEFFFFEEEEEEEE
```

#### SALIDA ESPERADA

```
t=0 llamada OK → CERRADO
t=1 llamada OK → CERRADO
t=2 llamada FALLA → CERRADO
t=3 llamada FALLA → CERRADO
t=4 llamada FALLA → ABIERTO
t=5 rechazada → ABIERTO
t=6 rechazada → ABIERTO
t=7 rechazada → ABIERTO
t=8 llamada OK → CERRADO
t=9 llamada OK → CERRADO
t=10 llamada FALLA → CERRADO
t=11 llamada OK → CERRADO
t=12 llamada OK → CERRADO
t=13 llamada OK → CERRADO
Llamadas al servicio: 11, rechazadas sin llamar: 3
```

### Ejemplo: La prueba falla

#### ENTRADA

```
3 2
FFFEFEFEFE
```

#### SALIDA ESPERADA

```
t=0 llamada FALLA → CERRADO
t=1 llamada FALLA → CERRADO
t=2 llamada FALLA → ABIERTO
t=3 rechazada → ABIERTO
t=4 llamada FALLA → ABIERTO
t=5 rechazada → ABIERTO
t=6 llamada FALLA → ABIERTO
t=7 rechazada → ABIERTO
t=8 llamada OK → CERRADO
t=9 llamada OK → CERRADO
Llamadas al servicio: 7, rechazadas sin llamar: 3
```

## Programación Segura en Java

### 9. Guardar contraseñas con PBKDF2

Difícil · Programación Segura en Java · [apuntesdam.com/subject/prog-servicios/topic/programacion-segura](https://apuntesdam.com/subject/prog-servicios/topic/programacion-segura)

Nunca se guarda una contraseña, ni cifrada: se guarda un hash lento con sal. Implementa guardar, que genera una sal aleatoria de 16 bytes con `SecureRandom`, deriva el hash con el método `derive` (PBKDF2-HMAC-SHA256, 32 bytes) usando `ITERACIONES` y devuelve el texto «pbkdf2\$iteraciones\$sal\$hash», con la sal y el hash en Base64. Implementa también verificar, que separa las cuatro partes (devuelve false si el formato no es correcto), vuelve a derivar el hash de la contraseña introducida con la misma sal y las mismas iteraciones guardadas y lo compara en tiempo constante con `MessageDigest.isEqual`. El main comprueba el formato, que dos guardados de la misma contraseña son distintos y la verificación; la orden `VECTOR` calcula un hash con una sal fija para comprobar que `derive` se usa bien.

## Código de partida (java)

```
import java.security.*;
import java.security.spec.*;
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class Main {
    static final int ITERACIONES = 120_000;

    // Deriva 32 bytes de la contraseña con PBKDF2-HMAC-SHA256
    static byte[] derivar(String clave, byte[] sal, int iteraciones) throws GeneralSecurityException {
        PBEKeySpec spec = new PBEKeySpec(clave.toCharArray(), sal, iteraciones, 256);
        return SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256").generateSecret(spec).getEncoded();
    }

    // Devuelve "pbkdf2$iteraciones$salBase64$hashBase64" con una sal aleatoria de 16 bytes
    static String guardar(String clave) throws GeneralSecurityException {
        // TODO
        return clave;
    }

    // Comprueba una contraseña contra lo guardado, en tiempo constante
    static boolean verificar(String clave, String guardado) throws GeneralSecurityException {
        // TODO
        return guardado.equals(clave);
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().split(" ", 3);
            switch (t[0]) {
                case "VECTOR" -> { // VECTOR salHex clave: resultado reproducible con 1000
                    iteraciones
                    byte[] sal = HexFormat.of().parseHex(t[1]);
                    System.out.println("Hash: " + HexFormat.of().formatHex(derivar(t[2], sal, 1000)));
                }
                case "PROBAR" -> { // PROBAR claveGuardada claveIntroducida
                    String g1 = guardar(t[1]), g2 = guardar(t[1]);
                    String[] p = g1.split("\\$");
                    System.out.println("Formato: " + p.length + " partes, algoritmo " + p[0] + ", " +
                        (p.length > 1 ? p[1] : "?") + " iteraciones, sal de "
                        + Base64.getDecoder().decode(p.length > 2 ? p[2] : "").length + " bytes, hash
                        de "
                        + Base64.getDecoder().decode(p.length > 3 ? p[3] : "").length + " bytes");
                    System.out.println("Dos guardados de la misma clave son distintos: " +
                        !g1.equals(g2));
                    System.out.println("La clave guardada no aparece en el texto: " +
                        !g1.contains(t[1]));
                    System.out.println("Verificar «" + t[2] + "»: " + verificar(t[2], g1));
                }
            }
        }
    }
}
```

### Ejemplo: Vector fijo

#### ENTRADA

VECTOR 000102030405060708090a0b0c0d0e0f patata

#### SALIDA ESPERADA

Hash:  
e5816b525c317a12ddd9f411a9304dc0e131d02a179b24a02  
834f989d4613578

### Ejemplo: Clave correcta

#### ENTRADA

PROBAR Clave.Segura-2025 Clave.Segura-2025

#### SALIDA ESPERADA

Formato: 4 partes, algoritmo pbkdf2, 120000 iteraciones, sal de 16 bytes, hash de 32 bytes  
Dos guardados de la misma clave son distintos: true  
La clave guardada no aparece en el texto: true  
Verificar «Clave.Segura-2025»: true

## 10. Tokens firmados con HMAC (como un JWT)

Muy difícil · Programación Segura en Java · [apuntesdam.com/subject/prog-servicios/topic/programacion-segura](https://apuntesdam.com/subject/prog-servicios/topic/programacion-segura)

Tras iniciar sesión, muchas APIs entregan un token que el cliente envía en cada petición. Para que no se pueda falsificar, va firmado. Implementa crear: el token son tres partes separadas por puntos, cada una en Base64URL sin relleno: la cabecera {"alg":"HS256","typ":"JWT"}, los datos {"sub":"usuario","rol":"rol","exp":instante} (exactamente con ese formato, sin espacios) y la firma HMAC-SHA256 de «cabecera.datos» con la clave del servidor (usa el método hmac). Implementa verificar: si no hay tres partes o no se pueden decodificar, «INVÁLIDO: formato»; si la firma no coincide (compárala en tiempo constante), «INVÁLIDO: firma»; si ahora  $\geq$  exp, «INVÁLIDO: caducado»; si todo está bien, «VÁLIDO usuario rol». Comprueba la firma antes de mirar los datos. El método campo, ya hecho, lee un valor del JSON de los datos.

## Código de partida (java)

```
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class Main {
    static final Base64.Encoder B64 = Base64.getUrlEncoder().withoutPadding();
    static final Base64.Decoder D64 = Base64.getUrlDecoder();

    static byte[] hmac(byte[] clave, String datos) throws GeneralSecurityException {
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(clave, "HmacSHA256"));
        return mac.doFinal(datos.getBytes(StandardCharsets.UTF_8));
    }

    // Token: base64url(cabecera) + "." + base64url(datos) + "." + base64url(HMAC de las dos primeras partes)
    // La cabecera es {"alg":"HS256","typ":"JWT"} y los datos {"sub":"usuario","rol":"rol","exp":instante}
    static String crear(byte[] clave, String usuario, String rol, long exp) throws GeneralSecurityException {
        // TODO
        return "";
    }

    // Devuelve "VÁLIDO usuario rol", o "INVÁLIDO: motivo" (formato, firma o caducado)
    static String verificar(byte[] clave, String token, long ahora) throws GeneralSecurityException {
        // TODO
        return "VÁLIDO";
    }

    // Extrae un campo de un JSON plano y sencillo como el de los datos
    static String campo(String json, String nombre) {
        int i = json.indexOf("\"" + nombre + "\"");
        if (i < 0) return null;
        int ini = i + nombre.length() + 3;
        if (json.charAt(ini) == '"') return json.substring(ini + 1, json.indexOf('"', ini + 1));
        int fin = ini;
        while (fin < json.length() && (Character.isDigit(json.charAt(fin)) || json.charAt(fin) == '-'))
            fin++;
        return json.substring(ini, fin);
    }

    public static void main(String[] args) throws Exception {
        byte[] clave = "clave-secreta-del-servidor-de-32b".getBytes(StandardCharsets.UTF_8);
        byte[] otraClave = "otra-clave-que-no-es-la-buena-32".getBytes(StandardCharsets.UTF_8);
        Scanner sc = new Scanner(System.in);
        String usuario = sc.next(), rol = sc.next();
        long exp = sc.nextLong();
        String token = crear(clave, usuario, rol, exp);
        System.out.println("Token: " + token);
        while (sc.hasNext()) {
            String prueba = sc.next();
            long ahora = sc.nextLong();
            String t = switch (prueba) {
                case "normal" -> token;
                case "otra-clave" -> crear(otraClave, usuario, rol, exp);
                case "cambiar-rol" -> { // el atacante edita los datos y deja la firma original
                    String[] p = token.split("\\.");
                    String datos = new String(D64.decode(p.length > 1 ? p[1] : ""), StandardCharsets.UTF_8).replace(rol, "admin");
                    yield p[0] + "." + B64.encodeToString(datos.getBytes(StandardCharsets.UTF_8)) + "." + (p.length > 2 ? p[2] : "");
                }
                default -> "esto.no.es-un-token.valido";
            };
            System.out.println(prueba + " en t=" + ahora + ": " + verificar(clave, t, ahora));
        }
    }
}
```



### **Formato de la entrada**

- Línea 1: número de hilos (1 a 8). Resto: el texto.

**Palabras vacías (no salen en «Más frecuentes»)**

## Palabras

de, la, que, el, en, y, a, los, se, del, las, un, por, con, no, una, su, para, es, al, lo, como

### Ejemplo: Un texto con tres hilos

#### ENTRADA

```
3
La programación concurrente permite que varios
hilos trabajen a la vez.
Cada hilo cuenta las palabras de su bloque de
líneas.
Si los hilos comparten un mapa sin protección, se
pierden datos.
Por eso cada hilo usa un mapa propio y devuelve
el resultado.
El hilo principal espera con get() y suma los
mapas parciales.
Con un hilo o con ocho hilos, el resultado debe
ser el mismo.
La concurrencia bien hecha no cambia el
resultado: solo el tiempo.
```

#### SALIDA ESPERADA

```
Hilos: 3 · Líneas: 7
Bloque [1-3]: 32 palabras
Bloque [4-5]: 23 palabras
Bloque [6-7]: 24 palabras
Palabras: 79 · Distintas: 56
Más frecuentes:
4 hilo
3 hilos
3 resultado
2 cada
2 mapa
```

### Ejemplo: Número de hilos no válido

#### ENTRADA

```
12
Un texto cualquiera
```

#### SALIDA ESPERADA

```
Número de hilos no válido: debe estar entre 1 y 8
```

#### Código de partida (java)

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Main {
    static final Set<String> VACIAS = Set.of("de", "la", "que", "el", "en", "y", "a", "los", "se", "del",
        "las", "un",
        "por", "con", "no", "una", "su", "para", "es", "al", "lo", "como");

    /** Lo que hace cada hilo: contar las palabras de su bloque en un mapa propio, sin compartir nada. */
    static Map<String, Integer> contar(List<String> lineas) {
        // TODO
        return new HashMap<>();
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        // TODO: lee el número de hilos y el texto, reparte los bloques en un pool de hilos y une los
        resultados
    }
}
```

## 12. Servidor de órdenes con sockets e hilos

Muy difícil · Java · 100 minutos · [apuntesdam.com/ejercicios/java/servidor-de-ordenes-con-sockets](https://apuntesdam.com/ejercicios/java/servidor-de-ordenes-con-sockets)

Muchos servicios de Internet (SMTP, FTP, Redis, POP3) usan protocolos de texto: el cliente envía una orden por línea y el servidor responde con otra línea que empieza por un código ( `OK` , `ERR` ). Son fáciles de depurar, porque se pueden probar a mano con telnet o netcat.

Vas a escribir un servidor de este tipo en Java. Escucha en un puerto, acepta conexiones y atiende a cada cliente en un hilo distinto, para que un cliente lento no bloquee a los demás. Tiene un pequeño almacén clave-valor compartido por todas las conexiones, al estilo de Redis, así que tiene que ser seguro para varios hilos.

Para poder probarlo sin dos programas, el mismo `main` arranca el servidor en un hilo y hace de cliente: lee las órdenes de la entrada, las envía por el socket y escribe lo que contesta el servidor. Una línea `---` cierra la conexión y abre otra, como si llegara un cliente nuevo.

## Requisitos

- El servidor escucha en `localhost` en un puerto libre (puerto 0) y atiende cada conexión en un hilo nuevo. Lee líneas en UTF-8 y responde a cada una con una línea. Cuando recibe `ADIOS` responde y cierra la conexión.
- Protocolo (las órdenes se reconocen en mayúsculas o minúsculas): `SUMA a b` → `OK suma` con enteros de hasta 9 cifras y signo opcional, o `ERR números no válidos`; `MAYUS texto` → `OK TEXTO`; `INVIERTE texto` → `OK otxtet` (sin texto, `ERR falta el texto`); `GUARDA clave valor` (el valor puede tener espacios) → `OK guardado` u `OK sustituido` (antes: `valor`), y `ERR uso: GUARDA clave valor` si falta algo; `LEE clave` → `OK valor` o `ERR no existe clave`; `CUENTA` → `OK número de claves`; `ADIOS` → `OK adiós`. Una línea vacía es `ERR línea vacía` y cualquier otra orden `ERR orden desconocida`.
- El cliente conecta al empezar y escribe `== Cliente N conectado`. Por cada línea de la entrada (incluidas las vacías) la envía, espera la respuesta y escribe `> línea` (solo `>` si la línea está vacía) y `< respuesta`. Una línea `---` cierra la conexión actual y abre una nueva (con su mensaje de conexión); después de `ADIOS`, la siguiente línea también abre una conexión nueva.
- Al terminar la entrada, el cliente cierra su conexión, se cierra el servidor y se escribe `Servidor cerrado · claves guardadas: N`.
- El almacén es común a todas las conexiones: lo que guarda un cliente lo puede leer el siguiente.

## Formato de la entrada

- Una orden del protocolo por línea; `---` abre una conexión nueva.

### Ejemplo: Dos clientes que comparten el almacén

#### ENTRADA

```
SUMA 20 22
MAYUS hola, mundo
INVIERTE ñandú
GUARDA curso 2º DAM
GUARDA curso 1º DAW
CUENTA
ADIOS
LEE curso
GUARDA tutor Ana Gil
---
LEE tutor
LEE profe
CUENTA
```

#### SALIDA ESPERADA

```
== Cliente 1 conectado
> SUMA 20 22
< OK 42
> MAYUS hola, mundo
< OK HOLA, MUNDO
> INVIERTE ñandú
< OK údnañ
> GUARDA curso 2º DAM
< OK guardado
> GUARDA curso 1º DAW
< OK sustituido (antes: 2º DAM)
> CUENTA
< OK 1
> ADIOS
< OK adiós
== Cliente 2 conectado
> LEE curso
< OK 1º DAW
> GUARDA tutor Ana Gil
< OK guardado
== Cliente 3 conectado
> LEE tutor
< OK Ana Gil
> LEE profe
< ERR no existe profe
> CUENTA
< OK 2
Servidor cerrado · claves guardadas: 2
```

### Ejemplo: Errores del protocolo

#### ENTRADA

```
SUMA 1
SUMA uno dos
MAYUS

GUARDA solo
BORRA todo
suma -5 3
```

#### SALIDA ESPERADA

```
== Cliente 1 conectado
> SUMA 1
< ERR números no válidos
> SUMA uno dos
< ERR números no válidos
> MAYUS
< ERR falta el texto
>
< ERR línea vacía
> GUARDA solo
< ERR uso: GUARDA clave valor
> BORRA todo
< ERR orden desconocida
> suma -5 3
< OK -2
Servidor cerrado · claves guardadas: 0
```

#### Código de partida (java)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.Map;
import java.util.Scanner;
import java.util.concurrent.ConcurrentHashMap;

public class Main {
    static final Map<String, String> almacen = new ConcurrentHashMap<>();

    /** La respuesta del protocolo a una línea. */
    static String responder(String linea) {
        // TODO: SUMA, MAYUS, INVIERTE, GUARDA, LEE, CUENTA y ADIOS
        return "ERR sin hacer";
    }

    /** Atiende a un cliente: una respuesta por línea hasta ADIOS o hasta que cierre la conexión. */
    static void atender(Socket socket) {
        // TODO
    }

    public static void main(String[] args) throws Exception {
        // TODO: arranca el servidor en un hilo, conecta el cliente y envía las líneas de la entrada
    }
}
```

## 13. Almacén de contraseñas seguro con PBKDF2 y sal

Difícil · Java · 70 minutos · [apuntesdam.com/ejercicios/java/almacen-de-contrasenas-con-pbkdf2](https://apuntesdam.com/ejercicios/java/almacen-de-contrasenas-con-pbkdf2)

Guardar contraseñas en texto plano, o con un hash rápido como MD5 o SHA-256 sin más, es uno de los fallos de seguridad más graves y más frecuentes. Si alguien roba la base de datos, con un hash rápido puede probar miles de millones de contraseñas por segundo.

La forma correcta es usar una función de derivación lenta a propósito, como PBKDF2, bcrypt o Argon2, con una sal aleatoria distinta para cada usuario. La sal hace que dos usuarios con la misma contraseña tengan hashes distintos (y que no sirvan las tablas precalculadas), y las miles de iteraciones hacen que cada intento cueste tiempo.

El inicio de sesión también tiene sus reglas: comparar los hashes en tiempo constante, bloquear la cuenta tras varios fallos seguidos y responder lo mismo si el usuario no existe que si la contraseña es incorrecta, para que nadie pueda averiguar qué usuarios hay. Como la sal es aleatoria, el programa nunca escribe hashes: solo el resultado de cada operación.

## Requisitos

- Cada línea es una orden con sus argumentos separados por espacios (la orden en mayúsculas o minúsculas; las líneas vacías se ignoran). Antes de responder se escribe `> ORDEN usuario:` (la orden en mayúsculas y solo el primer argumento, nunca la contraseña) y después la respuesta en la misma línea. Una orden desconocida responde `orden desconocida`.
- `REGISTRA usuario contraseña`: el usuario ya existe; o, si la contraseña no cumple la política, `contraseña rechazada, motivo` con el primer motivo: `debe tener al menos 10 caracteres`, `debe tener mayúsculas, minúsculas y números`, `no puede contener el nombre de usuario` (sin distinguir mayúsculas); si no, guarda una sal aleatoria de 16 bytes y el hash PBKDF2WithHmacSHA256 (10.000 iteraciones, 256 bits) y responde `registrado`.
- `ENTRA usuario contraseña`: si el usuario no existe o la contraseña no es correcta, `usuario o contraseña incorrectos`; al tercer fallo seguido, `usuario o contraseña incorrectos; cuenta bloqueada tras 3 fallos`; con la cuenta bloqueada, `cuenta bloqueada` aunque la contraseña sea buena; si es correcta, `bienvenido` y el contador de fallos vuelve a 0.
- `CAMBIA usuario actual nueva`: no se puede cambiar la contraseña si el usuario no existe, está bloqueado o la actual no es correcta; después, la política (`contraseña rechazada, motivo`) y `la nueva contraseña debe ser distinta`; si todo va bien, nueva sal y nuevo hash, y `contraseña cambiada`.
- `DESBLOQUEA usuario` → `desbloqueada` (y fallos a 0) o `no existe`. `COMPARA u1 u2` → `¿mismo hash que u2? sí/no` o `no existe alguno de los usuarios`. `INFO usuario` → `PBKDF2WithHmacSHA256, 10000 iteraciones, sal de 16 bytes, hash de 32 bytes` (con `,` `bloqueada` al final si lo está) o `no existe`.
- Una orden con un número de argumentos que no es el suyo responde `uso: ...` con su formato (`uso: REGISTRA usuario contraseña`, `uso: ENTRA usuario contraseña`, `uso: CAMBIA usuario actual nueva`).
- Las contraseñas se comparan con `MessageDigest.isEqual`, nunca con `Arrays.equals` ni comparando textos.

## Formato de la entrada

- Una orden por línea: `REGISTRA`, `ENTRA`, `CAMBIA`, `DESBLOQUEA`, `COMPARA` o `INFO`.

### Ejemplo: Registro, inicio de sesión y bloqueo

#### ENTRADA

```
REGISTRA ana Corta1
REGISTRA ana sinmayusculas123
REGISTRA ana MiAnaClave2026
REGISTRA ana Gatos.Tigres.2026
REGISTRA eva Gatos.Tigres.2026
REGISTRA ana OtraClave99999
COMPARA ana eva
ENTRA ana gatos.tigres.2026
ENTRA nadie Gatos.Tigres.2026
ENTRA ana Gatos.Tigres.2026
ENTRA eva mal
ENTRA eva mal
ENTRA eva mal
ENTRA eva Gatos.Tigres.2026
INFO eva
```

#### SALIDA ESPERADA

```
> REGISTRA ana: contraseña rechazada, debe tener
al menos 10 caracteres
> REGISTRA ana: contraseña rechazada, debe tener
mayúsculas, minúsculas y números
> REGISTRA ana: contraseña rechazada, no puede
contener el nombre de usuario
> REGISTRA ana: registrado
> REGISTRA eva: registrado
> REGISTRA ana: el usuario ya existe
> COMPARA ana: ¿mismo hash que eva? no
> ENTRA ana: usuario o contraseña incorrectos
> ENTRA nadie: usuario o contraseña incorrectos
> ENTRA ana: bienvenido
> ENTRA eva: usuario o contraseña incorrectos
> ENTRA eva: usuario o contraseña incorrectos
> ENTRA eva: usuario o contraseña incorrectos;
cuenta bloqueada tras 3 fallos
> ENTRA eva: cuenta bloqueada
> INFO eva: PBKDF2WithHmacSHA256, 10000
iteraciones, sal de 16 bytes, hash de 32 bytes,
bloqueada
```

## Ejemplo: Cambio de contraseña y desbloqueo

### ENTRADA

```
REGISTRA luis Montañas.2026
CAMBIA luis Equivocada1 Nueva.Clave.27
CAMBIA luis Montañas.2026 Montañas.2026
CAMBIA luis Montañas.2026 corta
CAMBIA luis Montañas.2026 Nueva.Clave.27
ENTRA luis Montañas.2026
ENTRA luis Nueva.Clave.27
DESBLOQUEA luis
INFO luis
```

### SALIDA ESPERADA

```
> REGISTRA luis: registrado
> CAMBIA luis: no se puede cambiar la contraseña
> CAMBIA luis: la nueva contraseña debe ser distinta
> CAMBIA luis: contraseña rechazada, debe tener al menos 10 caracteres
> CAMBIA luis: contraseña cambiada
> ENTRAR luis: usuario o contraseña incorrectos
> ENTRAR luis: bienvenido
> DESBLOQUEA luis: desbloqueada
> INFO luis: PBKDF2WithHmacSHA256, 10000 iteraciones, sal de 16 bytes, hash de 32 bytes
```

### Código de partida (java)

```
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

public class Main {
    static final String ALGORITMO = "PBKDF2WithHmacSHA256";
    static final int ITERACIONES = 10_000; // en producción, cientos de miles: aquí menos para que las pruebas vayan rápido
    static final int MAX_FALLOS = 3;

    static class Cuenta {
        byte[] sal;
        byte[] hash;
        int fallos;
        boolean bloqueada;
    }

    static final Map<String, Cuenta> cuentas = new HashMap<>();
    static final SecureRandom azar = new SecureRandom();

    static byte[] derivar(String contrasena, byte[] sal) throws Exception {
        PBEKeySpec spec = new PBEKeySpec(contrasena.toCharArray(), sal, ITERACIONES, 256);
        return SecretKeyFactory.getInstance(ALGORITMO).generateSecret(spec).getEncoded();
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        // TODO: REGISTRA, ENTRAR, CAMBIA, DESBLOQUEA, COMPARAR e INFO
    }
}
```

# Soluciones

## 1. Coordinador de procesos

Difícil · Programación Concurrente · [apuntesdam.com/subject/prog-servicios/topic/concurrencia](https://apuntesdam.com/subject/prog-servicios/topic/concurrencia)

Solución (java)

```
import java.io.*;
import java.nio.file.Path;
import java.util.*;

public class Main {
    static boolean esPrimo(long n) {
        if (n < 2) return false;
        for (long d = 2; d * d <= n; d++) if (n % d == 0) return false;
        return true;
    }

    static long contar(long desde, long hasta) {
        long c = 0;
        for (long i = desde; i <= hasta; i++) if (esPrimo(i)) c++;
        return c;
    }

    public static void main(String[] args) throws Exception {
        // Modo trabajador: el proceso hijo cuenta su tramo y escribe el resultado
        if (args.length == 3 && args[0].equals("trabajador")) {
            System.out.println(contar(Long.parseLong(args[1]), Long.parseLong(args[2])));
            return;
        }
        Scanner sc = new Scanner(System.in);
        long limite = sc.nextLong();
        int procesos = sc.nextInt();

        // El mismo java que nos ejecuta y el mismo classpath, para lanzar esta clase en otro proceso
        String java = Path.of(System.getProperty("java.home"), "bin", "java").toString();
        String classpath = System.getProperty("java.class.path");
        long tamano = limite / procesos;
        List<Process> hijos = new ArrayList<>();
        List<long[]> tramos = new ArrayList<>();
        for (int i = 0; i < procesos; i++) {
            long desde = i * tamano + 1;
            long hasta = i == procesos - 1 ? limite : (i + 1) * tamano; // el último se queda con el resto
            tramos.add(new long[]{desde, hasta});
            ProcessBuilder pb = new ProcessBuilder(java, "-cp", classpath, "Main", "trabajador",
                String.valueOf(desde), String.valueOf(hasta));
            pb.redirectErrorStream(true);
            hijos.add(pb.start()); // primero se lanzan todos
        }

        long total = 0;
        for (int i = 0; i < procesos; i++) {
            Process hijo = hijos.get(i);
            long n;
            try (BufferedReader br = new BufferedReader(new InputStreamReader(hijo.getInputStream()))) {
                n = Long.parseLong(br.readLine().trim());
            }
            hijo.waitFor();
            total += n;
            System.out.println("Proceso " + (i + 1) + " [" + tramos.get(i)[0] + ", " + tramos.get(i)[1] +
                "]: " + n + " primos");
        }
        System.out.println("Total: " + total + " primos hasta " + limite);
    }
}
```

`ProcessBuilder` lanza un programa como proceso independiente. Para que el hijo sea esta misma clase se usa el mismo ejecutable de Java ( `java.home` ) y el mismo classpath, y se le pasan los argumentos «trabajador desde hasta», que hacen que el `main` entre en el modo trabajador.

Lanzar todos los procesos antes de leer ninguno es lo que permite que trabajen a la vez: si se lanzara uno, se esperara su resultado y después se lanzara el siguiente, irían uno tras otro. La salida estándar de cada hijo se lee con `getInputStream()` y `waitFor()` espera a que termine.

El reparto usa la división entera: cada tramo mide `limite / procesos` y el último llega hasta el límite, quedándose con lo que sobra.

## 2. Planificador de CPU: FCFS, SJF y Round Robin

Muy difícil · Programación Concurrente · [apuntesdam.com/subject/prog-servicios/topic/concurrencia](http://apuntesdam.com/subject/prog-servicios/topic/concurrencia)

Solución (java)

```
import java.util.*;

public class Main {
    record Proceso(String id, int llegada, int rafaga) { }

    // Procesos por orden de llegada (a igualdad, el de la entrada: sort es estable)
    static List<Proceso> porLlegada(List<Proceso> ps) {
        List<Proceso> l = new ArrayList<>(ps);
        l.sort(Comparator.comparingInt(Proceso::llegada));
        return l;
    }

    // Devuelve la línea de tiempo: una letra (id) por unidad de tiempo, o '-' si la CPU está libre
    static String fcfs(List<Proceso> ps) {
        StringBuilder sb = new StringBuilder();
        for (Proceso p : porLlegada(ps)) {
            while (sb.length() < p.llegada()) sb.append('-'); // CPU libre hasta que llega
            sb.append(p.id().repeat(p.rafaga()));
        }
        return sb.toString();
    }

    static String sjf(List<Proceso> ps) {
        StringBuilder sb = new StringBuilder();
        List<Proceso> pendientes = new ArrayList<>(ps);
        while (!pendientes.isEmpty()) {
            int t = sb.length();
            Proceso elegido = null;
            for (Proceso p : pendientes) {
                if (p.llegada() <= t && (elegido == null || p.rafaga() < elegido.rafaga())) elegido = p;
            }
            if (elegido == null) { sb.append('-'); continue; } // nadie ha llegado todavía
            sb.append(elegido.id().repeat(elegido.rafaga())); // no expropiativo: hasta el final
            pendientes.remove(elegido);
        }
        return sb.toString();
    }

    static String roundRobin(List<Proceso> ps, int quantum) {
        StringBuilder sb = new StringBuilder();
        List<Proceso> llegadas = porLlegada(ps);
        Map<Proceso, Integer> resto = new HashMap<>();
        for (Proceso p : ps) resto.put(p, p.rafaga());
        Deque<Proceso> cola = new ArrayDeque<>();
        int siguiente = 0;
        while (siguiente < llegadas.size() || !cola.isEmpty()) {
            while (siguiente < llegadas.size() && llegadas.get(siguiente).llegada() <= sb.length())
                cola.add(llegadas.get(siguiente++));
            if (cola.isEmpty()) { sb.append('-'); continue; }
            Proceso p = cola.poll();
            int turno = Math.min(quantum, resto.get(p));
            sb.append(p.id().repeat(turno));
            resto.put(p, resto.get(p) - turno);
            // los que han llegado durante el turno entran antes que el expulsado
            while (siguiente < llegadas.size() && llegadas.get(siguiente).llegada() <= sb.length())
                cola.add(llegadas.get(siguiente++));
            if (resto.get(p) > 0) cola.add(p);
        }
        return sb.toString();
    }

    // Calcula y muestra los tiempos de espera a partir de la línea de tiempo
    static void informe(String nombre, String linea, List<Proceso> ps) {
        StringBuilder sb = new StringBuilder(nombre + ": " + linea + " |");
        double suma = 0;
        for (Proceso p : ps) {
```

```

        int fin = linea.lastIndexOf(p.id().charAt(0)) + 1;
        int espera = fin - p.llegada() - p.rafaga();
        suma += espera;
        sb.append(" ").append(p.id()).append("=").append(espera);
    }
    sb.append(String.format(Locale.US, " | media %.2f", suma / ps.size()));
    System.out.println(sb);
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int quantum = sc.nextInt();
    List<Proceso> ps = new ArrayList<>();
    while (sc.hasNext()) ps.add(new Proceso(sc.next(), sc.nextInt(), sc.nextInt()));
    informe("FCFS", fcfs(ps), ps);
    informe("SJF ", sjf(ps), ps);
    informe("RR  ", roundRobin(ps, quantum), ps);
}
}

```

En los tres algoritmos la longitud de la línea de tiempo es el reloj: `sb.length()` dice en qué instante estamos, y una CPU libre se representa añadiendo un «-» y avanzando una unidad.

FCFS ordena por llegada (el `sort` de Java es estable, así que respeta el orden de la entrada en los empates) y ejecuta cada proceso entero. SJF, cada vez que la CPU queda libre, busca entre los que ya han llegado el de ráfaga más corta; como compara con `<`, en un empate se queda el primero.

Round Robin usa una cola: el proceso de la cabeza se ejecuta como mucho un quantum y, si le queda ráfaga, vuelve al final. El orden de esas dos operaciones decide el resultado: primero entran en la cola los procesos que han llegado durante el turno y después el expulsado.

### 3. Suma en paralelo con hilos

Difícil · Hilos y Multithreading en Java · [apuntesdam.com/subject/prog-servicios/topic/hilos](https://apuntesdam.com/subject/prog-servicios/topic/hilos)

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);
        int hilos = sc.nextInt();
        List<Long> lista = new ArrayList<>();
        while (sc.hasNextLong()) lista.add(sc.nextLong());
        long[] datos = lista.stream().mapToLong(Long::longValue).toArray();

        long[] parciales = new long[hilos]; // cada hilo escribe solo en su posición
        Thread[] trabajadores = new Thread[hilos];
        int base = datos.length / hilos, extra = datos.length % hilos;
        for (int i = 0; i < hilos; i++) {
            final int id = i;
            final int desde = i * base + Math.min(i, extra); // los primeros bloques tienen uno más
            final int hasta = desde + base + (i < extra ? 1 : 0);
            trabajadores[i] = new Thread(() -> {
                long suma = 0;
                for (int k = desde; k < hasta; k++) suma += datos[k];
                parciales[id] = suma;
            });
            trabajadores[i].start();
        }
        for (Thread t : trabajadores) t.join(); // esperar a que terminen todos

        long total = 0;
        for (int i = 0; i < hilos; i++) {
            System.out.println("Hilo " + i + ": " + parciales[i]);
            total += parciales[i];
        }
        System.out.println("Total: " + total);
    }
}
```

Cada bloque mide `n / hilos` elementos, y los `n % hilos` primeros se llevan uno más. El inicio del bloque `i` es `i * base + min(i, extra)`: los bloques anteriores ocupan `base` cada uno más uno extra por cada uno de ellos que lo tenía.

Los hilos no comparten nada que puedan pisarse: leen el array (que nadie modifica) y cada uno escribe en su propia posición de `parciales`, así que no hace falta sincronización. Lo que sí es imprescindible es `join()`: sin él, el `main` podría leer los parciales antes de que los hilos los calculen.

Las variables que usa la lambda tienen que ser `final` (o efectivamente finales); por eso se copian `i`, `desde` y `hasta` en variables nuevas en cada vuelta.

## 4. Productor-consumidor con un búfer acotado

Muy difícil · Hilos y Multithreading en Java · [apuntesdam.com/subject/prog-servicios/topic/hilos](http://apuntesdam.com/subject/prog-servicios/topic/hilos)

Solución (java)

```
import java.util.*;

class BufferAcotado {
    private final int[] datos;
    private int inicio = 0, cantidad = 0;

    BufferAcotado(int capacidad) {
        datos = new int[capacidad];
    }

    // Espera mientras el búfer está lleno
    public synchronized void poner(int v) throws InterruptedException {
        while (cantidad == datos.length) wait(); // lleno: a esperar
        datos[(inicio + cantidad) % datos.length] = v; // el hueco tras el último
        cantidad++;
        notifyAll(); // puede haber alguien esperando para sacar
    }

    // Espera mientras el búfer está vacío
    public synchronized int sacar() throws InterruptedException {
        while (cantidad == 0) wait(); // vacío: a esperar
        int v = datos[inicio];
        inicio = (inicio + 1) % datos.length;
        cantidad--;
        notifyAll(); // puede haber alguien esperando para poner
        return v;
    }
}

public class Main {
    static final int FIN = -1; // «píldora venenosa»: avisa al consumidor de que no
    hay más

    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);
        int capacidad = sc.nextInt(), productores = sc.nextInt(), porProductor = sc.nextInt();
        BufferAcotado buffer = new BufferAcotado(capacidad);
        int[] consumidosDe = new int[productores];
        long[] suma = new long[1];

        List<Thread> hilos = new ArrayList<>();
        for (int p = 0; p < productores; p++) {
            final int id = p;
            hilos.add(new Thread(() -> {
                try {
                    // Cada valor codifica su productor: id * 1000 + número
                    for (int k = 1; k <= porProductor; k++) buffer.poner(id * 1000 + k);
                } catch (InterruptedException e) { Thread.currentThread().interrupt(); }
            }));
        }
        Thread consumidor = new Thread(() -> {
            try {
                int v;
                while ((v = buffer.sacar()) != FIN) {
                    consumidosDe[v / 1000]++;
                    suma[0] += v % 1000;
                }
            } catch (InterruptedException e) { Thread.currentThread().interrupt(); }
        });
        consumidor.start();
        for (Thread h : hilos) h.start();
        for (Thread h : hilos) h.join();
        buffer.poner(FIN);
        consumidor.join();
    }
}
```

```
    for (int p = 0; p < productores; p++) System.out.println("Productor " + p + ": " +  
        consumidosDe[p] + " elementos consumidos");  
    System.out.println("Suma de los números: " + suma[0]);  
}  
}
```

`synchronized` garantiza que solo un hilo a la vez modifica el búfer. Dentro, `wait()` suelta el cerrojo y duerme al hilo hasta que otro llame a `notifyAll()`; por eso se avisa cada vez que el estado cambia, para despertar a quien esté esperando el cambio contrario.

La condición se comprueba con `while` y no con `if`: al despertar, otro hilo puede haber llenado (o vaciado) el búfer antes, y hay que volver a mirar. Con `notifyAll` se despierta a todos, productores y consumidor, y cada uno comprueba si puede seguir.

El array circular evita mover elementos: el primero está en `inicio`, el hueco libre en `(inicio + cantidad) % capacidad`, y los dos índices dan la vuelta al llegar al final con el módulo.

## 5. Servidor de un protocolo de texto

Difícil · Sockets en Java · [apuntesdam.com/subject/prog-servicios/topic/sockets](http://apuntesdam.com/subject/prog-servicios/topic/sockets)

Solución (java)

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Main {
    // Atiende a UN cliente: lee órdenes línea a línea y responde a cada una
    static void atender(Socket cliente) throws IOException {
        BufferedReader in = new BufferedReader(new InputStreamReader(cliente.getInputStream(), "UTF-8"));
        PrintWriter out = new PrintWriter(new OutputStreamWriter(cliente.getOutputStream(), "UTF-8"),
            true);
        int atendidas = 0;
        String linea;
        while ((linea = in.readLine()) != null) {
            String[] p = linea.trim().split(" ", 2);
            String orden = p[0].toUpperCase(), resto = p.length > 1 ? p[1] : "";
            String respuesta;
            switch (orden) {
                case "SUMA" -> {
                    try {
                        long total = 0;
                        for (String n : resto.trim().split("\\s+")) total += Long.parseLong(n);
                        respuesta = "OK " + total;
                    } catch (NumberFormatException e) {
                        respuesta = "ERROR números no válidos";
                    }
                }
                case "MAYUS" -> respuesta = "OK " + resto.toUpperCase();
                case "INVERTIR" -> respuesta = "OK " + new StringBuilder(resto).reverse();
                case "CONTAR" -> respuesta = "OK " + atendidas;
                case "ADIOS" -> respuesta = "OK hasta luego";
                default -> respuesta = "ERROR orden desconocida: " + p[0];
            }
            atendidas++;
            out.println(respuesta); // autoflush: se envía al momento
            if (orden.equals("ADIOS")) break; // al salir, el main cierra el socket
        }
    }

    public static void main(String[] args) throws Exception {
        try (ServerSocket servidor = new ServerSocket(0)) { // puerto libre cualquiera
            Thread hiloServidor = new Thread(() -> {
                try (Socket c = servidor.accept()) {
                    atender(c);
                } catch (IOException e) {
                    System.out.println("Error del servidor: " + e.getMessage());
                }
            });
            hiloServidor.start();

            // Cliente: envía cada línea de la entrada y muestra la respuesta
            Scanner sc = new Scanner(System.in);
            try (Socket s = new Socket("localhost", servidor.getLocalPort());
                PrintWriter out = new PrintWriter(new OutputStreamWriter(s.getOutputStream(), "UTF-8"),
                    true);
                BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream(), "UTF-8"))) {
                while (sc.hasNextLine()) {
                    String orden = sc.nextLine();
                    out.println(orden);
                    String r;
                    try {
                        r = in.readLine();
                    } catch (IOException e) {
                        r = null; // el servidor cerró la conexión
                    }
                    System.out.println("> " + orden);
                }
            }
        }
    }
}
```

```

        System.out.println("< " + (r == null ? "(conexión cerrada)" : r));
        if (r == null || orden.equals("ADIOS")) break;
    }
    hiloServidor.join();
}
}
}

```

Un protocolo de texto línea a línea se implementa con un `BufferedReader` para leer órdenes y un `PrintWriter` con `autoFlush` para responder: sin el `true` del constructor, la respuesta se quedaría en el búfer y el cliente esperaría para siempre. Los dos se crean con UTF-8 para que las tildes y la ñ viajen bien.

Cada línea se parte en la orden y el resto con `split(" ", 2)`, y la orden se pasa a mayúsculas para que «suma» y «SUMA» sean lo mismo. El contador se incrementa después de responder, así CONTAR informa de las órdenes anteriores a ella. El bucle termina con ADIOS o cuando `readLine` devuelve `null` porque el cliente cerró la conexión.

## 6. Mensajes fiables sobre UDP

Muy difícil · Sockets en Java · [apuntesdam.com/subject/prog-servicios/topic/sockets](https://apuntesdam.com/subject/prog-servicios/topic/sockets)

Solución (java)

```
import java.io.*;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.util.*;

public class Main {
    static final int ESPERA_MS = 150, MAX_INTENTOS = 5;

    // ---- Servidor (ya hecho): pierde paquetes a propósito para simular una red mala
    static class Servidor extends Thread {
        final DatagramSocket ds;
        final List<String> recibidos = new ArrayList<>();
        int paquetes = 0, duplicados = 0, ultimoSeq = 0;

        Servidor() throws SocketException { ds = new DatagramSocket(0); }

        public void run() {
            byte[] buf = new byte[1024];
            try {
                while (true) {
                    DatagramPacket p = new DatagramPacket(buf, buf.length);
                    ds.receive(p);
                    paquetes++;
                    if (paquetes % 4 == 0) continue; // se pierde el paquete que llega
                    String msg = new String(p.getData(), 0, p.getLength(), StandardCharsets.UTF_8);
                    int seq = Integer.parseInt(msg.substring(0, msg.indexOf(':')));
                    String datos = msg.substring(msg.indexOf(':') + 1);
                    if (seq == ultimoSeq) duplicados++; // ya lo teníamos: era una retransmisión
                    else if (seq == ultimoSeq + 1) { ultimoSeq = seq; if (!datos.equals("FIN"))
                        recibidos.add(datos); }
                    if (paquetes % 3 == 0) continue; // se pierde el ACK de respuesta
                    byte[] ack = ("ACK " + seq).getBytes(StandardCharsets.UTF_8);
                    ds.send(new DatagramPacket(ack, ack.length, p.getSocketAddress()));
                    if (datos.equals("FIN")) return;
                }
            } catch (IOException e) { }
        }
    }

    // Envía un mensaje y espera su ACK; si no llega a tiempo, lo reenvía.
    // Devuelve el número de intentos, o -1 si se agotan.
    static int enviarFiable(DatagramSocket ds, SocketAddress destino, int seq, String texto) throws
        IOException {
        byte[] datos = (seq + ":" + texto).getBytes(StandardCharsets.UTF_8);
        byte[] buf = new byte[1024];
        String esperado = "ACK " + seq;
        for (int intento = 1; intento <= MAX_INTENTOS; intento++) {
            ds.send(new DatagramPacket(datos, datos.length, destino));
            long limite = System.currentTimeMillis() + ESPERA_MS;
            while (true) {
                long queda = limite - System.currentTimeMillis();
                if (queda <= 0) break; // se acabó el tiempo: reenviar
                ds.setSoTimeout((int) queda);
                DatagramPacket p = new DatagramPacket(buf, buf.length);
                try {
                    ds.receive(p);
                } catch (SocketTimeoutException e) {
                    break;
                }
                String r = new String(p.getData(), 0, p.getLength(), StandardCharsets.UTF_8);
                if (r.equals(esperado)) return intento;
                // un ACK de otro número de secuencia (llegó tarde): se ignora y se sigue esperando
            }
        }
        return -1;
    }
}
```

```

    }

    public static void main(String[] args) throws Exception {
        Servidor srv = new Servidor();
        srv.start();
        SocketAddress destino = new InetSocketAddress("localhost", srv.ds.getLocalPort());
        Scanner sc = new Scanner(System.in);
        List<String> mensajes = new ArrayList<>();
        while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) mensajes.add(l); }
        mensajes.add("FIN");
        try (DatagramSocket ds = new DatagramSocket()) {
            for (int i = 0; i < mensajes.size(); i++) {
                int intentos = enviarFiable(ds, destino, i + 1, mensajes.get(i));
                System.out.println("Mensaje " + (i + 1) + " «" + mensajes.get(i) + "»: "
                    + (intentos < 0 ? "sin respuesta" : intentos + (intentos == 1 ? " intento" : " intentos")));
            }
        }
        srv.join(2000);
        srv.ds.close();
        System.out.println("El servidor recibió en orden: " + String.join(" | ", srv.recibidos));
        System.out.println("Retransmisiones duplicadas descartadas: " + srv.duplicados);
    }
}

```

Parada y espera: se envía el mensaje y se espera su confirmación antes de mandar el siguiente. `setSoTimeout` hace que `receive` deje de esperar al cabo de un tiempo lanzando `SocketTimeoutException`; ese es el momento de reenviar el mismo mensaje, con el mismo número de secuencia.

Si llega un ACK que no es el esperado, no se puede volver a esperar `ESPERA_MS` completos: el tiempo de espera se mide desde el envío. Por eso se calcula un instante límite y, en cada vuelta, se espera solo lo que queda hasta él.

El número de secuencia es lo que hace que el protocolo funcione cuando se pierde un ACK: el cliente reenvía un mensaje que ya había llegado, y el servidor lo reconoce como duplicado y no lo guarda dos veces.

## 7. Una API REST con HttpServer

Difícil · Desarrollo de Servicios en Red · [apuntesdam.com/subject/prog-servicios/topic/servicios-red-java](http://apuntesdam.com/subject/prog-servicios/topic/servicios-red-java)

Solución (java)

```
import com.sun.net.httpserver.*;
import java.io.*;
import java.net.*;
import java.net.http.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicInteger;

public class Main {
    static final Map<Integer, String> tareas = new ConcurrentSkipListMap<>(); // id → texto, ordenadas
    static final AtomicInteger siguienteId = new AtomicInteger(1);

    static void responder(HttpExchange ex, int estado, String cuerpo) throws IOException {
        byte[] b = cuerpo.getBytes(StandardCharsets.UTF_8);
        ex.getResponseHeaders().set("Content-Type", "application/json; charset=utf-8");
        ex.sendResponseHeaders(estado, b.length == 0 ? -1 : b.length);
        if (b.length > 0) ex.getResponseBody().write(b);
        ex.close();
    }

    static String json(int id, String texto) {
        return "{\"id\":\"" + id + "\",\"texto\":\"" + texto.replace("\\", "\\\\").replace("\"", "\\\"") +
            "\"}\"";
    }

    static String error(String mensaje) {
        return "{\"error\":\"" + mensaje + "\"}";
    }

    // Atiende /tareas y /tareas/{id}
    static void manejar(HttpExchange ex) throws IOException {
        String metodo = ex.getRequestMethod();
        String[] partes = ex.getRequestURI().getPath().split("/"); // "", "tareas" y quizá el id

        if (partes.length == 2) { // la colección: /tareas
            switch (metodo) {
                case "GET" -> {
                    List<String> lista = new ArrayList<>();
                    tareas.forEach((id, texto) -> lista.add(json(id, texto)));
                    responder(ex, 200, "[" + String.join(",", lista) + "]");
                }
                case "POST" -> {
                    String texto = new String(ex.getRequestBody().readAllBytes(),
                        StandardCharsets.UTF_8).trim();
                    if (texto.isEmpty()) {
                        responder(ex, 400, error("texto vacío"));
                        return;
                    }
                    int id = siguienteId.getAndIncrement();
                    tareas.put(id, texto);
                    ex.getResponseHeaders().set("Location", "/tareas/" + id);
                    responder(ex, 201, json(id, texto));
                }
                default -> responder(ex, 405, "");
            }
            return;
        }

        int id; // un elemento: /tareas/{id}
        try {
            id = Integer.parseInt(partes[2]);
        } catch (NumberFormatException e) {
            responder(ex, 400, error("id no válido"));
            return;
        }
    }
}
```

```

    }
    if (!metodo.equals("GET") && !metodo.equals("DELETE")) {
        responder(ex, 405, "");
        return;
    }
    String texto = tareas.get(id);
    if (texto == null) {
        responder(ex, 404, error("no existe la tarea " + id));
    } else if (metodo.equals("GET")) {
        responder(ex, 200, json(id, texto));
    } else {
        tareas.remove(id);
        responder(ex, 204, "");
    }
}

public static void main(String[] args) throws Exception {
    HttpServer srv = HttpServer.create(new InetSocketAddress("localhost", 0), 0);
    srv.createContext("/tareas", Main::manejar);
    srv.start();
    String base = "http://localhost:" + srv.getAddress().getPort();

    HttpClient cliente = HttpClient.newHttpClient();
    Scanner sc = new Scanner(System.in);
    while (sc.hasNextLine()) {
        String[] t = sc.nextLine().trim().split(" ", 3);          // MÉTODO ruta [cuerpo]
        HttpRequest.BodyPublisher cuerpo = t.length > 2
            ? HttpRequest.BodyPublishers.ofString(t[2]) : HttpRequest.BodyPublishers.noBody();
        HttpRequest req = HttpRequest.newBuilder(URI.create(base + t[1])).method(t[0],
            cuerpo).build();
        HttpResponse<String> r = cliente.send(req, HttpResponse.BodyHandlers.ofString());
        System.out.println(t[0] + " " + t[1] + " → " + r.statusCode() + (r.body().isEmpty() ? "" : "
            " + r.body()));
    }
    srv.stop(0);
}
}

```

Un mismo contexto atiende la colección ( `/tareas` ) y cada elemento ( `/tareas/{id}` ); partiendo la ruta por «/» se distinguen por el número de partes. Dentro de cada caso, el método HTTP decide la operación, que es la idea central de REST: la URL nombra el recurso y el método dice qué hacer con él.

Cada error tiene su código: 400 si la petición está mal (texto vacío o id que no es un número), 404 si el recurso no existe y 405 si el método no tiene sentido para esa URL. Crear devuelve 201 con la cabecera Location que dice dónde está el nuevo recurso, y borrar devuelve 204, que significa «hecho, sin contenido».

El mapa es un `ConcurrentSkipListMap` porque `HttpServer` puede atender peticiones en varios hilos, y además mantiene las tareas ordenadas por id; el contador es un `AtomicInteger` por la misma razón.

## 8. Disyuntor (circuit breaker) para un servicio inestable

Muy difícil · Desarrollo de Servicios en Red · [apuntesdam.com/subject/prog-servicios/topic/servicios-red-java](https://apuntesdam.com/subject/prog-servicios/topic/servicios-red-java)

Solución (java)

```
import java.util.*;

class CircuitBreaker {
    enum Estado { CERRADO, ABIERTO, SEMIABIERTO }

    private final int umbralFallos;      // fallos seguidos para abrir
    private final int espera;           // instantes que permanece abierto
    private Estado estado = Estado.CERRADO;
    private int fallosSeguidos = 0;
    private int abiertoDesde = 0;

    CircuitBreaker(int umbralFallos, int espera) {
        this.umbralFallos = umbralFallos;
        this.espera = espera;
    }

    Estado estado() { return estado; }

    // ¿Se puede llamar al servicio en el instante t? (puede pasar de ABIERTO a SEMIABIERTO)
    boolean permitir(int t) {
        if (estado != Estado.ABIERTO) return true;
        if (t - abiertoDesde >= espera) {    // ya ha esperado bastante: se deja pasar una llamada de prueba
            estado = Estado.SEMIABIERTO;
            return true;
        }
        return false;
    }

    // Se informa del resultado de una llamada que se permitió
    void registrar(boolean exito, int t) {
        if (estado == Estado.SEMIABIERTO) {
            if (exito) {
                estado = Estado.CERRADO;
                fallosSeguidos = 0;
            } else {
                abrir(t);                // un solo fallo basta para volver a abrir
            }
            return;
        }
        if (exito) fallosSeguidos = 0;
        else if (++fallosSeguidos >= umbralFallos) abrir(t);
    }

    private void abrir(int t) {
        estado = Estado.ABIERTO;
        abiertoDesde = t;
        fallosSeguidos = 0;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircuitBreaker cb = new CircuitBreaker(sc.nextInt(), sc.nextInt());
        String resultados = sc.next();    // una letra por instante: E = el servicio responde,
        F = falla
        int llamadas = 0, rechazadas = 0;
        for (int t = 0; t < resultados.length(); t++) {
            String linea;
            if (cb.permite(t)) {
                llamadas++;
                boolean ok = resultados.charAt(t) == 'E';
                cb.registrar(ok, t);
                linea = (ok ? "llamada OK " : "llamada FALLA");
            } else {

```

```

        rechazadas++;
        linea = "rechazada  ";
    }
    System.out.println("t=" + t + " " + linea + " → " + cb.estado());
}
System.out.println("Llamadas al servicio: " + llamadas + ", rechazadas sin llamar: " +
    rechazadas);
}
}

```

Es una máquina de tres estados. `permitir` solo tiene trabajo en ABIERTO: rechaza las llamadas hasta que pasa el tiempo de espera y entonces deja pasar una de prueba cambiando a SEMIABIERTO. `registrar` decide qué hacer con el resultado según el estado en que se hizo la llamada.

En CERRADO se cuentan los fallos seguidos (un éxito pone el contador a 0) y se abre al llegar al umbral. En SEMIABIERTO no hay segunda oportunidad: un éxito cierra el circuito y un fallo lo abre de nuevo. El método privado `abrir` reúne lo que hay que hacer en los dos sitios donde se abre.

## 9. Guardar contraseñas con PBKDF2

Difícil · Programación Segura en Java · [apuntesdam.com/subject/prog-servicios/topic/programacion-segura](https://apuntesdam.com/subject/prog-servicios/topic/programacion-segura)

Solución (java)

```
import java.security.*;
import java.security.spec.*;
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class Main {
    static final int ITERACIONES = 120_000;

    // Deriva 32 bytes de la contraseña con PBKDF2-HMAC-SHA256
    static byte[] derivar(String clave, byte[] sal, int iteraciones) throws GeneralSecurityException {
        PBEKeySpec spec = new PBEKeySpec(clave.toCharArray(), sal, iteraciones, 256);
        return SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256").generateSecret(spec).getEncoded();
    }

    // Devuelve "pbkdf2$iteraciones$salBase64$hashBase64" con una sal aleatoria de 16 bytes
    static String guardar(String clave) throws GeneralSecurityException {
        byte[] sal = new byte[16];
        new SecureRandom().nextBytes(sal);
        byte[] hash = derivar(clave, sal, ITERACIONES);
        Base64.Encoder b64 = Base64.getEncoder();
        return "pbkdf2$" + ITERACIONES + "$" + b64.encodeToString(sal) + "$" + b64.encodeToString(hash);
    }

    // Comprueba una contraseña contra lo guardado, en tiempo constante
    static boolean verificar(String clave, String guardado) throws GeneralSecurityException {
        String[] p = guardado.split("\\$");
        if (p.length != 4 || !p[0].equals("pbkdf2")) return false;
        try {
            int iteraciones = Integer.parseInt(p[1]);
            byte[] sal = Base64.getDecoder().decode(p[2]);
            byte[] esperado = Base64.getDecoder().decode(p[3]);
            byte[] calculado = derivar(clave, sal, iteraciones); // misma sal y mismas iteraciones
            return MessageDigest.isEqual(esperado, calculado);
        } catch (IllegalArgumentException e) { // número o Base64 mal formados
            return false;
        }
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().split(" ", 3);
            switch (t[0]) {
                case "VECTOR" -> { // VECTOR salHex clave: resultado reproducible con 1000 iteraciones
                    byte[] sal = HexFormat.of().parseHex(t[1]);
                    System.out.println("Hash: " + HexFormat.of().formatHex(derivar(t[2], sal, 1000)));
                }
                case "PROBAR" -> { // PROBAR claveGuardada claveIntroducida
                    String g1 = guardar(t[1]), g2 = guardar(t[1]);
                    String[] p = g1.split("\\$");
                    System.out.println("Formato: " + p.length + " partes, algoritmo " + p[0] + ", " +
                        (p.length > 1 ? p[1] : "?") + " iteraciones, sal de " +
                        Base64.getDecoder().decode(p.length > 2 ? p[2] : "").length + " bytes, hash de " +
                        Base64.getDecoder().decode(p.length > 3 ? p[3] : "").length + " bytes");
                    System.out.println("Dos guardados de la misma clave son distintos: " +
                        !g1.equals(g2));
                    System.out.println("La clave guardada no aparece en el texto: " +
                        !g1.contains(t[1]));
                    System.out.println("Verificar «" + t[2] + "»: " + verificar(t[2], g1));
                }
            }
        }
    }
}
```

```
}
```

La sal aleatoria hace que dos usuarios con la misma contraseña tengan hashes distintos y que las tablas precalculadas no sirvan; por eso se genera con `SecureRandom` (un generador criptográfico, no `Random`) y se guarda junto al hash, porque hará falta para verificar. Las muchas iteraciones hacen que cada intento cueste tiempo, lo que frena los ataques de fuerza bruta.

Verificar no es «descifrar»: se toma la contraseña introducida, se deriva otra vez con la misma sal y las mismas iteraciones que se guardaron, y se comparan los dos hashes. Guardar las iteraciones en el texto permite aumentarlas en el futuro sin invalidar las contraseñas antiguas.

`MessageDigest.isEqual` compara en tiempo constante: tarda lo mismo aunque los primeros bytes no coincidan, así un atacante no puede ir adivinando el hash byte a byte midiendo cuánto tarda la respuesta.

## 10. Tokens firmados con HMAC (como un JWT)

Muy difícil · Programación Segura en Java · [apuntesdam.com/subject/prog-servicios/topic/programacion-segura](https://apuntesdam.com/subject/prog-servicios/topic/programacion-segura)

Solución (java)

```
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class Main {
    static final Base64.Encoder B64 = Base64.getUrlEncoder().withoutPadding();
    static final Base64.Decoder D64 = Base64.getUrlDecoder();

    static byte[] hmac(byte[] clave, String datos) throws GeneralSecurityException {
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(clave, "HmacSHA256"));
        return mac.doFinal(datos.getBytes(StandardCharsets.UTF_8));
    }

    static String b64(String texto) {
        return B64.encodeToString(texto.getBytes(StandardCharsets.UTF_8));
    }

    // Token: base64url(cabecera) + "." + base64url(datos) + "." + base64url(HMAC de las dos primeras partes)
    // La cabecera es {"alg":"HS256","typ":"JWT"} y los datos {"sub":"usuario","rol":"rol","exp":instante}
    static String crear(byte[] clave, String usuario, String rol, long exp) throws GeneralSecurityException {
        String cabecera = b64("{\"alg\":\"HS256\",\"typ\":\"JWT\"}");
        String datos = b64("{\"sub\":\"" + usuario + "\",\"rol\":\"" + rol + "\",\"exp\":\"" + exp + "\"}");
        String firma = B64.encodeToString(hmac(clave, cabecera + "." + datos));
        return cabecera + "." + datos + "." + firma;
    }

    // Devuelve "VÁLIDO usuario rol", o "INVÁLIDO: motivo" (formato, firma o caducado)
    static String verificar(byte[] clave, String token, long ahora) throws GeneralSecurityException {
        String[] p = token.split("\\.");
        if (p.length != 3) return "INVÁLIDO: formato";
        byte[] firma;
        String datos;
        try {
            D64.decode(p[0]);
            datos = new String(D64.decode(p[1]), StandardCharsets.UTF_8);
            firma = D64.decode(p[2]);
        } catch (IllegalArgumentException e) {
            return "INVÁLIDO: formato";
        }
        // Primero la firma: hasta que no se comprueba, nada de lo que dicen los datos es fiable
        if (!MessageDigest.isEqual(firma, hmac(clave, p[0] + "." + p[1]))) return "INVÁLIDO: firma";
        if (ahora >= Long.parseLong(campo(datos, "exp"))) return "INVÁLIDO: caducado";
        return "VÁLIDO " + campo(datos, "sub") + " " + campo(datos, "rol");
    }

    // Extrae un campo de un JSON plano y sencillo como el de los datos
    static String campo(String json, String nombre) {
        int i = json.indexOf("\"" + nombre + "\":");
        if (i < 0) return null;
        int ini = i + nombre.length() + 3;
        if (json.charAt(ini) == '"') return json.substring(ini + 1, json.indexOf('"', ini + 1));
        int fin = ini;
        while (fin < json.length() && (Character.isDigit(json.charAt(fin)) || json.charAt(fin) == '-'))
            fin++;
        return json.substring(ini, fin);
    }

    public static void main(String[] args) throws Exception {
        byte[] clave = "clave-secreta-del-servidor-de-32b".getBytes(StandardCharsets.UTF_8);
        byte[] otraClave = "otra-clave-que-no-es-la-buena-32".getBytes(StandardCharsets.UTF_8);
    }
}
```

```

Scanner sc = new Scanner(System.in);
String usuario = sc.next(), rol = sc.next();
long exp = sc.nextLong();
String token = crear(clave, usuario, rol, exp);
System.out.println("Token: " + token);
while (sc.hasNext()) {
    String prueba = sc.next();
    long ahora = sc.nextLong();
    String t = switch (prueba) {
        case "normal" -> token;
        case "otra-clave" -> crear(otraClave, usuario, rol, exp);
        case "cambiar-rol" -> { // el atacante edita los datos y deja la firma
            original
            String[] p = token.split("\\.");
            String datos = new String(D64.decode(p.length > 1 ? p[1] : ""),
                StandardCharsets.UTF_8).replace(rol, "admin");
            yield p[0] + "." + B64.encodeToString(datos.getBytes(StandardCharsets.UTF_8)) + "." +
                (p.length > 2 ? p[2] : "");
        }
        default -> "esto.no.es-un-token.valido";
    };
    System.out.println(prueba + " en t=" + ahora + ": " + verificar(clave, t, ahora));
}
}
}

```

El token no está cifrado: cualquiera puede decodificar las dos primeras partes y leer los datos. Lo que lo protege es la firma, un HMAC calculado con una clave que solo conoce el servidor. Si alguien cambia un solo carácter de los datos (por ejemplo, el rol), la firma ya no coincide, y no puede calcular la buena sin la clave.

Por eso el orden de las comprobaciones importa: primero el formato, después la firma y solo entonces se confía en los datos para mirar la caducidad. La firma se compara con `MessageDigest.isEqual`, en tiempo constante, por la misma razón que las contraseñas.

Base64URL sin relleno cambia `+` y `/` por `-` y `_` y quita los `=` finales, para que el token pueda viajar en una URL o en una cabecera sin escapar nada.

## 11. Contador de palabras en paralelo con un pool de hilos

Difícil · Java · 70 minutos · [apuntesdam.com/ejercicios/java/contador-de-palabras-con-hilos](https://apuntesdam.com/ejercicios/java/contador-de-palabras-con-hilos)

Solución (java)

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Main {
    static final Set<String> VACIAS = Set.of("de", "la", "que", "el", "en", "y", "a", "los", "se", "del",
        "las", "un",
        "por", "con", "no", "una", "su", "para", "es", "al", "lo", "como");

    /** Lo que hace cada hilo: contar las palabras de su bloque en un mapa propio, sin compartir nada. */
    static Map<String, Integer> contar(List<String> lineas) {
        Map<String, Integer> cuenta = new HashMap<>();
        for (String linea : lineas) {
            for (String palabra : linea.toLowerCase().split("[^a-záéíóúñ]+")) {
                if (!palabra.isEmpty()) cuenta.merge(palabra, 1, Integer::sum);
            }
        }
        return cuenta;
    }

    static int suma(Map<String, Integer> cuenta) {
        int total = 0;
        for (int n : cuenta.values()) total += n;
        return total;
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        String primera = sc.hasNextLine() ? sc.nextLine().trim() : "";
        if (!primera.matches("[1-8]")) {
            System.out.println("Número de hilos no válido: debe estar entre 1 y 8");
            return;
        }
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            if (!linea.isBlank()) lineas.add(linea);
        }
        if (lineas.isEmpty()) {
            System.out.println("No hay texto que analizar");
            return;
        }

        // Nunca más hilos que líneas; los bloques, lo más iguales posible
        int hilos = Math.min(Integer.parseInt(primera), lineas.size());
        ExecutorService pool = Executors.newFixedThreadPool(hilos);
        List<Future<Map<String, Integer>>> resultados = new ArrayList<>();
        List<String> bloques = new ArrayList<>();
        int inicio = 0;
        for (int i = 0; i < hilos; i++) {
            int tamaño = lineas.size() / hilos + (i < lineas.size() % hilos ? 1 : 0);
            List<String> bloque = lineas.subList(inicio, inicio + tamaño);
            resultados.add(pool.submit(() -> contar(bloque)));
            bloques.add(tamaño == 1 ? "[" + (inicio + 1) + "]" : "[" + (inicio + 1) + "-" + (inicio + tamaño) + "]");
            inicio += tamaño;
        }

        // Se unen los resultados en el hilo principal, en el orden de los bloques
        Map<String, Integer> total = new HashMap<>();
```

```

List<String> parciales = new ArrayList<>();
for (int i = 0; i < hilos; i++) {
    Map<String, Integer> parcial = resultados.get(i).get(); // get() espera a que termine ese hilo
    parciales.add(" Bloque " + bloques.get(i) + ": " + suma(parcial) + " palabras");
    parcial.forEach((palabra, n) -> total.merge(palabra, n, Integer::sum));
}
pool.shutdown();

System.out.println("Hilos: " + hilos + " · Líneas: " + lineas.size());
parciales.forEach(System.out::println);
System.out.println("Palabras: " + suma(total) + " · Distintas: " + total.size());

List<Map.Entry<String, Integer>> lista = new ArrayList<>();
for (Map.Entry<String, Integer> e : total.entrySet()) if (!VACIAS.contains(e.getKey()))
    lista.add(e);
lista.sort((a, b) -> !a.getValue().equals(b.getValue()) ? b.getValue() - a.getValue() :
    a.getKey().compareTo(b.getKey()));
System.out.println("Más frecuentes:");
for (int i = 0; i < Math.min(5, lista.size()); i++) {
    System.out.printf("%4d %s\n", lista.get(i).getValue(), lista.get(i).getKey());
}
}
}

```

Cada hilo trabaja sobre su propio bloque y su propio mapa, y devuelve el resultado en lugar de escribirlo en una variable compartida. Sin estado compartido no hay condiciones de carrera, así que no hacen falta `synchronized`, cerrojos ni `ConcurrentHashMap`.

`Callable` es la versión de `Runnable` que devuelve un valor, y `Future.get()` es el punto de encuentro: bloquea hasta que la tarea termina. Recorrer los `Future` en orden hace que los parciales se impriman siempre igual, aunque los hilos acaben en cualquier orden.

La suma de recuentos es asociativa y conmutativa: da igual cómo se reparta el texto y en qué orden se combinen los trozos, el total es el mismo. Esa propiedad es la que permite paralelizar este problema (y la que hay que comprobar antes de paralelizar cualquier otro).

El pool limita los hilos a los pedidos (y a las líneas que hay) y los reutiliza. Crear un `Thread` a mano por cada bloque funcionaría con 8 bloques, pero no con 10.000.

## 12. Servidor de órdenes con sockets e hilos

Muy difícil · Java · 100 minutos · [apuntesdam.com/ejercicios/java/servidor-de-ordenes-con-sockets](https://apuntesdam.com/ejercicios/java/servidor-de-ordenes-con-sockets)

Solución (java)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.Map;
import java.util.Scanner;
import java.util.concurrent.ConcurrentHashMap;

public class Main {
    // Compartido por todas las conexiones: cada cliente se atiende en su propio hilo
    static final Map<String, String> almacen = new ConcurrentHashMap<>();

    /** La respuesta del protocolo a una línea. */
    static String responder(String linea) {
        String[] p = linea.trim().split("\\s+", 2);
        String orden = p[0].toUpperCase();
        String resto = p.length > 1 ? p[1] : "";
        switch (orden) {
            case "" -> { return "ERR línea vacía"; }
            case "SUMA" -> {
                String[] n = resto.split("\\s+");
                if (n.length != 2 || !n[0].matches("-?\\d{1,9}") || !n[1].matches("-?\\d{1,9}")) return
                    "ERR números no válidos";
                return "OK " + (Long.parseLong(n[0]) + Long.parseLong(n[1]));
            }
            case "MAYUS" -> { return resto.isEmpty() ? "ERR falta el texto" : "OK " +
                resto.toUpperCase(); }
            case "INVIERTE" -> { return resto.isEmpty() ? "ERR falta el texto" : "OK " + new
                StringBuilder(resto).reverse(); }
            case "GUARDA" -> {
                String[] kv = resto.split("\\s+", 2);
                if (kv.length != 2) return "ERR uso: GUARDA clave valor";
                String anterior = almacen.put(kv[0], kv[1]);
                return anterior == null ? "OK guardado" : "OK sustituido (antes: " + anterior + ")";
            }
            case "LEE" -> {
                String valor = almacen.get(resto);
                return valor == null ? "ERR no existe " + resto : "OK " + valor;
            }
            case "CUENTA" -> { return "OK " + almacen.size(); }
            case "ADIOS" -> { return "OK adiós"; }
            default -> { return "ERR orden desconocida"; }
        }
    }

    /** Atiende a un cliente: una respuesta por línea hasta ADIOS o hasta que cierre la conexión. */
    static void atender(Socket socket) {
        try (socket;
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream(),
                StandardCharsets.UTF_8));
            PrintWriter out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(),
                StandardCharsets.UTF_8), true)) {
            String linea;
            while ((linea = in.readLine()) != null) {
                out.println(responder(linea));
                if (linea.trim().equalsIgnoreCase("ADIOS")) break;
            }
        } catch (IOException e) {
            System.out.println("Error con un cliente: " + e.getMessage());
        }
    }
}
```

```

public static void main(String[] args) throws Exception {
    // Puerto 0: el sistema elige uno libre
    ServerSocket servidor = new ServerSocket(0);
    Thread aceptador = new Thread(() -> {
        try {
            while (true) {
                Socket cliente = servidor.accept();
                new Thread(() -> atender(cliente)).start();
            }
        } catch (IOException e) {
            // El servidor se ha cerrado: se termina el bucle
        }
    });
    aceptador.start();

    // El cliente: cada «---» de la entrada abre una conexión nueva
    Scanner sc = new Scanner(System.in);
    int numero = 0;
    Socket conexion = null;
    BufferedReader in = null;
    PrintWriter out = null;
    while (sc.hasNextLine()) {
        String linea = sc.nextLine();
        if (conexion == null || linea.equals("---")) {
            if (conexion != null) conexion.close();
            conexion = new Socket("localhost", servidor.getLocalPort());
            in = new BufferedReader(new InputStreamReader(conexion.getInputStream(),
                StandardCharsets.UTF_8));
            out = new PrintWriter(new OutputStreamWriter(conexion.getOutputStream(),
                StandardCharsets.UTF_8), true);
            System.out.println("== Cliente " + (++numero) + " conectado");
            if (linea.equals("---")) continue;
        }
        out.println(linea);
        String respuesta = in.readLine();
        System.out.println(linea.isEmpty() ? ">" : "> " + linea);
        System.out.println("< " + (respuesta == null ? "(conexión cerrada)" : respuesta));
        if (respuesta != null && linea.trim().equalsIgnoreCase("ADIÓS")) {
            conexion.close();
            conexion = null;
        }
    }
    if (conexion != null) conexion.close();
    servidor.close();
    aceptador.join();
    System.out.println("Servidor cerrado · claves guardadas: " + almacen.size());
}
}

```

Separar el protocolo ( `responder` ) de la red ( `atender` ) es lo que hace manejable un servidor: la lógica se prueba como cualquier método y la parte de sockets queda reducida a leer una línea, responder y repetir.

Un hilo por cliente es el modelo más sencillo de servidor concurrente: mientras un hilo espera a que su cliente escriba, los demás siguen atendiendo. Para miles de conexiones se usa un pool de hilos o E/S no bloqueante, pero el protocolo sería el mismo.

El almacén es el único estado compartido, y por eso es un `ConcurrentHashMap`. Su `put` es atómico: con un `HashMap` y un `get` seguido de un `put`, dos clientes podrían creer a la vez que la clave era nueva.

Usar el puerto 0 hace que el sistema elija uno libre: el programa nunca falla porque el puerto esté ocupado. El cliente lo averigua con `getLocalPort()`, igual que en un test de integración real.

## 13. Almacén de contraseñas seguro con PBKDF2 y sal

Difícil · Java · 70 minutos · [apuntesdam.com/ejercicios/java/almacen-de-contrasenas-con-pbkdf2](https://apuntesdam.com/ejercicios/java/almacen-de-contrasenas-con-pbkdf2)

Solución (java)

```
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

public class Main {
    static final String ALGORITMO = "PBKDF2WithHmacSHA256";
    static final int ITERACIONES = 10_000; // en producción, cientos de miles: aquí menos para que las
    pruebas vayan rápido
    static final int MAX_FALLOS = 3;

    static class Cuenta {
        byte[] sal;
        byte[] hash;
        int fallos;
        boolean bloqueada;
    }

    static final Map<String, Cuenta> cuentas = new HashMap<>();
    static final SecureRandom azar = new SecureRandom();

    static byte[] derivar(String contrasena, byte[] sal) throws Exception {
        PBEKeySpec spec = new PBEKeySpec(contrasena.toCharArray(), sal, ITERACIONES, 256);
        return SecretKeyFactory.getInstance(ALGORITMO).generateSecret(spec).getEncoded();
    }

    /** null si la contraseña cumple la política, o el primer motivo por el que no. */
    static String politica(String usuario, String c) {
        if (c.length() < 10) return "debe tener al menos 10 caracteres";
        if (!c.matches(".*[A-Z].*") || !c.matches(".*[a-z].*") || !c.matches(".*\\d.*")) return "debe
            tener mayúsculas, minúsculas y números";
        if (c.toLowerCase().contains(usuario.toLowerCase())) return "no puede contener el nombre de
            usuario";
        return null;
    }

    static Cuenta nueva(String contrasena) throws Exception {
        Cuenta cuenta = new Cuenta();
        cuenta.sal = new byte[16];
        azar.nextBytes(cuenta.sal); // una sal distinta para cada contraseña
        cuenta.hash = derivar(contrasena, cuenta.sal);
        return cuenta;
    }

    /** Comprueba la contraseña en tiempo constante (MessageDigest.isEqual no corta en el primer byte
        distinto). */
    static boolean correcta(Cuenta cuenta, String contrasena) throws Exception {
        return MessageDigest.isEqual(cuenta.hash, derivar(contrasena, cuenta.sal));
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            String[] p = linea.split("\\s+");
            String orden = p[0].toUpperCase();
            System.out.print("> " + orden + (p.length > 1 ? " " + p[1] : "") + ": ");
            switch (orden) {
                case "REGISTRA" -> {
                    if (p.length != 3) { System.out.println("uso: REGISTRA usuario contraseña"); break; }
                    if (cuentas.containsKey(p[1])) { System.out.println("el usuario ya existe"); break; }
                    String motivo = politica(p[1], p[2]);
                }
            }
        }
    }
}
```

```

        if (motivo != null) { System.out.println("contraseña rechazada, " + motivo); break; }
        cuentas.put(p[1], nueva(p[2]));
        System.out.println("registrado");
    }
    case "ENTRA" -> {
        if (p.length != 3) { System.out.println("uso: ENTRA usuario contraseña"); break; }
        Cuenta cuenta = cuentas.get(p[1]);
        // El mismo mensaje si el usuario no existe: así no se puede averiguar qué usuarios
        hay
        if (cuenta == null) { System.out.println("usuario o contraseña incorrectos"); break; }
        if (cuenta.bloqueada) { System.out.println("cuenta bloqueada"); break; }
        if (correcta(cuenta, p[2])) {
            cuenta.fallos = 0;
            System.out.println("bienvenido");
        } else if (++cuenta.fallos >= MAX_FALLOS) {
            cuenta.bloqueada = true;
            System.out.println("usuario o contraseña incorrectos; cuenta bloqueada tras " +
                MAX_FALLOS + " fallos");
        } else {
            System.out.println("usuario o contraseña incorrectos");
        }
    }
    case "CAMBIA" -> {
        if (p.length != 4) { System.out.println("uso: CAMBIA usuario actual nueva"); break; }
        Cuenta cuenta = cuentas.get(p[1]);
        if (cuenta == null || cuenta.bloqueada || !correcta(cuenta, p[2])) {
            System.out.println("no se puede cambiar la contraseña"); break; }
        String motivo = politica(p[1], p[3]);
        if (motivo != null) { System.out.println("contraseña rechazada, " + motivo); break; }
        if (p[3].equals(p[2])) { System.out.println("la nueva contraseña debe ser distinta");
            break; }
        cuentas.put(p[1], nueva(p[3]));
        System.out.println("contraseña cambiada");
    }
    case "DESBLOQUEA" -> {
        Cuenta cuenta = p.length == 2 ? cuentas.get(p[1]) : null;
        if (cuenta == null) { System.out.println("no existe"); break; }
        cuenta.bloqueada = false;
        cuenta.fallos = 0;
        System.out.println("desbloqueada");
    }
    case "COMPARA" -> {
        Cuenta a = p.length == 3 ? cuentas.get(p[1]) : null, b = p.length == 3 ?
            cuentas.get(p[2]) : null;
        if (a == null || b == null) { System.out.println("no existe alguno de los usuarios");
            break; }
        System.out.println("¿mismo hash que " + p[2] + "? " + (MessageDigest.isEqual(a.hash,
            b.hash) ? "sí" : "no"));
    }
    case "INFO" -> {
        Cuenta cuenta = p.length == 2 ? cuentas.get(p[1]) : null;
        if (cuenta == null) { System.out.println("no existe"); break; }
        System.out.println(ALGORITMO + ", " + ITERACIONES + " iteraciones, sal de " +
            cuenta.sal.length
                + " bytes, hash de " + cuenta.hash.length + " bytes" + (cuenta.bloqueada ? ",
                bloqueada" : ""));
    }
    default -> System.out.println("orden desconocida");
}
}
}
}

```

La contraseña nunca se guarda: se guardan la sal y el resultado de PBKDF2. Para comprobar un intento se repite la derivación con la misma sal; si coincide, la contraseña era la correcta. Ni siquiera el administrador puede recuperar una contraseña, solo cambiarla.

La sal aleatoria explica el resultado de **COMPARA**: aunque dos usuarios elijan la misma contraseña, sus hashes son distintos. Sin sal, bastaría ver dos hashes iguales en la base de datos robada para saber que comparten contraseña.

Las iteraciones son un parámetro de coste: cada una multiplica el trabajo de quien intente adivinar contraseñas por fuerza bruta. Aquí son 10.000 para que las pruebas vayan rápidas; las guías actuales recomiendan cientos de miles para PBKDF2, o usar Argon2.

El bloqueo tras tres fallos y los mensajes que no distinguen entre usuario y contraseña cierran las otras dos vías de ataque: probar contraseñas contra una cuenta y averiguar qué cuentas existen.