



MEDAC

Instituto Oficial de Formación Profesional

Práctica PSP — Actividad tema3: Gestión de procesos 1

Autor: Alberto Agredano

1. Resumen y objetivo.....	2
2. Archivos entregados.....	2
3. Código fuente.....	2
Aleatorios.java.....	2
OrdenarNumeros.java.....	3
4. Compilación.....	5
5. Ejecución — ejemplos.....	5
6. Sobre interbloqueo (deadlock) y buen diseño.....	7
7. Javadoc y documentación del código.....	7
8. Mini-manual de uso y pruebas.....	8

1. Resumen y objetivo

Esta práctica tiene 3 partes: una aplicación que ordena números leídos por la entrada estándar (`ordenarNumeros`), otra que genera números aleatorios y los escribe por la salida estándar (`aleatorios`), y la elaboración de un pequeño manual con las pruebas realizadas (incluyendo una prueba con tubería `|`).

En este documento encontrarás:

- Código Java con Javadoc y comentarios.
- Instrucciones paso a paso para compilar y ejecutar.
- Ejemplos de ejecución y cómo usar la tubería.

2. Archivos entregados

- `Aleatorios.java` — genera N números aleatorios (por defecto 40) entre 0 y 100.
- `OrdenarNumeros.java` — lee números desde stdin, los ordena y los imprime en stdout.
- `Manual_PracticaPSP.txt` — mini-manual con pasos de prueba y capturas sugeridas.

3. Código fuente

`Aleatorios.java`

```
/**
 * Aleatorios.java
 * -----
 * Genera una cantidad de números aleatorios entre 0 y 100 y los escribe
 * en la salida estándar (separados por espacios). Se puede pasar como
 * argumento el número de valores a generar.
 *
 * Uso:
 * java Aleatorios      (genera 40 números)
 * java Aleatorios 100  (genera 100 números)
```

```

*
* Está diseñado para usarse con tuberías, por ejemplo:
* java Aleatorios | java OrdenarNumeros
*
* @author Alberto
* @version 1.0
*/
import java.util.Random;

public class Aleatorios {
    public static void main(String[] args) {
        int count = 40; // valor por defecto
        if (args.length >= 1) {
            try {
                int c = Integer.parseInt(args[0]);
                if (c > 0) count = c;
            } catch (NumberFormatException e) {
                System.err.println("Aviso: argumento inválido para count. Usando 40.");
            }
        }

        Random rnd = new Random();
        StringBuilder sb = new StringBuilder();

        for (int i = 0; i < count; i++) {
            int n = rnd.nextInt(101); // genera 0..100
            sb.append(n);
            if (i < count - 1) sb.append(' ');
        }

        // Escribimos una sola línea con todos los números y forzamos flush.
        System.out.println(sb.toString());
        System.out.flush();
    }
}

```

OrdenarNumeros.java

```

/**
 * OrdenarNumeros.java
 * -----
 * Lee de la entrada estándar una secuencia indeterminada de tokens y
 * extrae los que sean números enteros, los ordena de menor a mayor y
 * los escribe en la salida estándar (un número por línea).
 *
 * Uso:

```

```

* java OrdenarNumeros
*
* Ejemplo con tubería:
* java Aleatorios 50 | java OrdenarNumeros
*
* Comportamiento en casos especiales:
* - Ignora tokens no numéricos y escribe un aviso por stderr para cada
  token inválido.
* - Si no se reciben números, informa y termina sin bloqueo.
*
* @author Alberto
* @version 1.0
*/
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

public class OrdenarNumeros {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Integer> lista = new ArrayList<>();

        // Leemos todos los tokens disponibles en stdin.
        while (sc.hasNext()) {
            String token = sc.next();
            try {
                int v = Integer.parseInt(token);
                lista.add(v);
            } catch (NumberFormatException e) {
                // No numérico: lo ignoramos, pero indicamos por stderr.
                System.err.println("Aviso: token no numérico ignorado -> '" + token + "'");
            }
        }
        sc.close();

        // Si no hay números, lo indicamos y salimos.
        if (lista.isEmpty()) {
            System.out.println("No se recibieron números. Fin de ejecución.");
            return;
        }

        // Ordenamos y mostramos (un valor por línea para claridad).
        Collections.sort(lista);
        for (int n : lista) {
            System.out.println(n);
        }
        System.out.flush();
    }
}

```

```
}  
}
```

4. Compilación

Abre una terminal y sitúate en la carpeta donde están los `.java`. Luego:

```
javac Aleatorios.java OrdenarNumeros.java
```

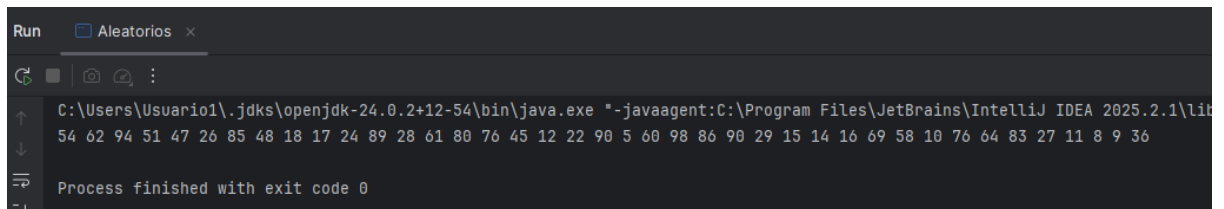
Esto genera `Aleatorios.class` y `OrdenarNumeros.class`.

5. Ejecución — ejemplos

1. Ejecutar `aleatorios` (40 por defecto):

```
java Aleatorios
```

salida: 23 0 55 100 7 ... (una línea con 40 números)



2. Ejecutar `ordenarNumeros` leyendo desde teclado (ejemplo suave):

escribir números y terminar con Ctrl+D (Unix) o Ctrl+Z + Enter (Windows)

```
java OrdenarNumeros
```

```
34 7 98 0 23
```

```
<Ctrl+D>
```

salida ordenada:

```
0
```

```
7
```

```
23
```

```
34
```

```
98
```

```
C:\Users\Usuario1\.jdk\openjdk
34
7
98
0
23
^D
0
7
23
34
98

Process finished with exit code
█
```

3. Uso de tubería — redirigir la salida de **Aleatorios** a **OrdenarNumeros**:

`java Aleatorios 50 | java OrdenarNumeros`
salida: 50 números ordenados, cada uno en una línea

```
PS C:\Users\Usuario1\IdeaProjects\Programacion_servicios_procesos\src\main\java> cd C:\Users\Usuario1\IdeaProjects\Programacion_servicios_procesos\src\main\java
PS C:\Users\Usuario1\IdeaProjects\Programacion_servicios_procesos\src\main\java> javac actividad1_tema3\Aleatorios.java actividad1_tema3\OrdenarNumeros.java
PS C:\Users\Usuario1\IdeaProjects\Programacion_servicios_procesos\src\main\java> java actividad1_tema3.Aleatorios 50 | java actividad1_tema3.OrdenarNumeros
1
3
6
13
18
20
25
26
26
32
32
34
36
37
37
38
44
44
47
49
53
54
57
60
62
65
65
67
71
71
```

4. Uso con archivo intermedio:

`java Aleatorios 100 > datos.txt`
`java OrdenarNumeros < datos.txt`

6. Sobre interbloqueo (deadlock) y buen diseño

- En este ejemplo **no hay riesgo de interbloqueo** entre los dos programas en condiciones normales porque:
 - **Aleatorios solo escribe** datos y termina (no espera entrada); una vez que termina, cierra su salida y el proceso consumidor recibe EOF cuando consume todos los bytes.
 - **OrdenarNumeros solo lee** de stdin hasta EOF y después realiza la ordenación y la salida.
- Posible bloqueo limitado: si el productor genera datos muy rápido y el consumidor es extremadamente lento, el kernel puede hacer que el **write** del productor bloquee (hasta que se vacíe el buffer de la tubería). Esto **no es un "deadlock"** entre procesos (no hay espera circular), sino bloqueo por presión de buffer; para esta práctica con 40–100 números es irrelevante.
- Buenas prácticas incluidas: forzamos **System.out.flush()** tras la escritura final para minimizar problemas con buffers si se redirige la salida.

7. Javadoc y documentación del código

Generar la documentación HTML con Javadoc:

```
javadoc -d doc Aleatorios.java OrdenarNumeros.java  
# Abre doc/index.html en tu navegador
```

```

PS C:\Users\Usuario1\IdeaProjects\Programacion_servicios_procesos\src\main\java\actividad1_tema3> javadoc -d doc Aleatorios.java OrdenarNumeros.java
Loading source file Aleatorios.java...
Loading source file OrdenarNumeros.java...
Constructing Javadoc information...
Creating destination directory: "doc\"
Building index for all the packages and classes...
Standard Doclet version 25+36-LTS
Building tree for all the packages and classes...
Generating doc\actividad1_tema3\Aleatorios.html...
Aleatorios.java:20: warning: no comment
public class Aleatorios {
      ^
Aleatorios.java:20: warning: use of default constructor, which does not provide a comment
public class Aleatorios {
      ^
Aleatorios.java:21: warning: no comment
    public static void main(String[] args) {
                  ^
Generating doc\actividad1_tema3\OrdenarNumeros.html...
OrdenarNumeros.java:28: warning: no comment
public class OrdenarNumeros {
      ^
OrdenarNumeros.java:28: warning: use of default constructor, which does not provide a comment
public class OrdenarNumeros {
      ^
OrdenarNumeros.java:29: warning: no comment
    public static void main(String[] args) {
                  ^
Generating doc\actividad1_tema3\package-summary.html...
Generating doc\actividad1_tema3\package-tree.html...
Generating doc\overview-tree.html...
Generating doc\allclasses-index.html...
Building index for all classes...
Generating doc\allpackages-index.html...
Generating doc\index-all.html...

```

8. Mini-manual de uso y pruebas

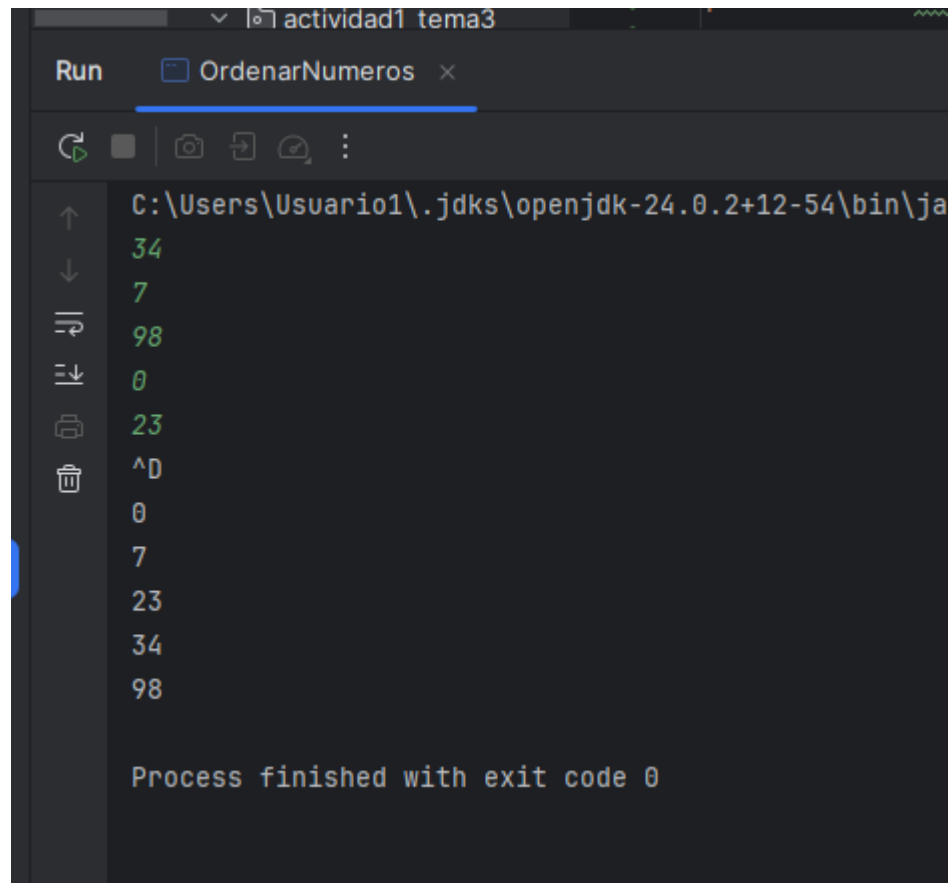
Para la práctica tendrás que entregar un pequeño manual con capturas. Sugerencia de contenido y capturas:

1. Compilación

- Captura: terminal mostrando `javac Aleatorios.java OrdenarNumeros.java` sin errores.

2. Ejecución de `Aleatorios`

- Captura: salida de `java Aleatorios 10` mostrando 10 números.

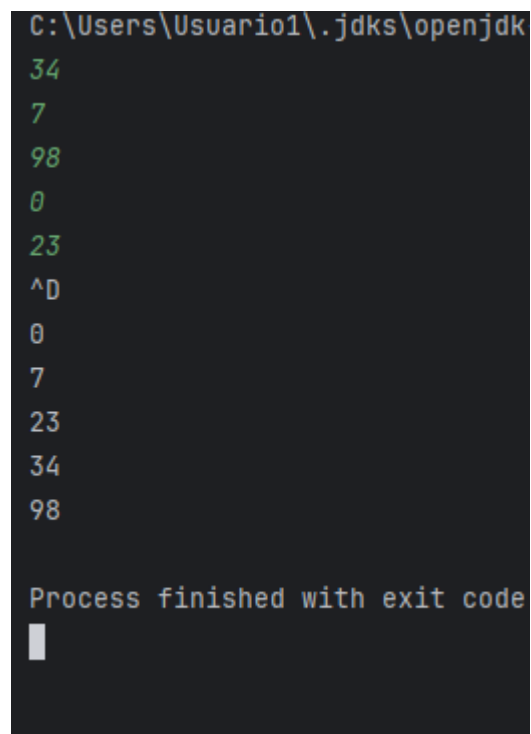


```
Run OrdenarNumeros x
C:\Users\Usuario1\.jdk\openjdk-24.0.2+12-54\bin\java
34
7
98
0
23
^D
0
7
23
34
98

Process finished with exit code 0
```

3. Ejecución de **OrdenarNumeros** con entrada manual

- Captura: introducir algunos números y la salida ordenada.



```
C:\Users\Usuario1\.jdk\openjdk
34
7
98
0
23
^D
0
7
23
34
98

Process finished with exit code 0
```

4. Ejecución con tubería (prueba requerida)

- Captura: `java Aleatorios 50 | java OrdenarNumeros` mostrando la salida ordenada.

```
PS C:\Users\Usuariol\IdeaProjects\Programacion_servicios_procesos\src\main\java> cd C:\Users\Usuariol\IdeaProjects\Programacion_servicios_procesos\src\main\java
PS C:\Users\Usuariol\IdeaProjects\Programacion_servicios_procesos\src\main\java> javac actividad1_tema3\Aleatorios.java actividad1_tema3\OrdenarNumeros.java
PS C:\Users\Usuariol\IdeaProjects\Programacion_servicios_procesos\src\main\java> java actividad1_tema3.Aleatorios 50 | java actividad1_tema3.OrdenarNumeros
1
3
6
13
18
20
25
26
32
32
34
36
37
37
38
44
44
47
49
53
54
57
60
62
65
65
67
71
71
```

5. Javadoc

- Captura: abrir `doc/index.html` en un navegador.



En cada captura añade una breve explicación debajo: qué comando ejecutaste, qué hace y por qué la prueba valida el requisito.