

# Orientación a Objetos II

## 2025

Explicación de práctica  
Semana del 12 de mayo



FACULTAD DE INFORMATICA



UNIVERSIDAD  
NACIONAL  
DE LA PLATA

# Ejercicios de la semana

## En el cuadernillo de Patrones

- Ejercicio 22/22b: Monitoreo de línea de producción
- Ejercicio 23: Acceso bajo demanda

## En el cuadernillo de Refactoring - **refactoring to patterns**

- Ejercicio 5 - Facturación de llamadas
- Ejercicio 6 - Árboles binarios

## En el cuadernillo de Frameworks

- Ejercicio 1: SingleThreadTCPFramework

# Ejercicio 22 - Monitoreo de línea de producción

*Una empresa necesita desarrollar un sistema para monitorear su línea de producción, en la cual se usa un tanque mezclador/calentador. El tanque posee un motor que mueve paletas internas y un calentador eléctrico; con esto se puede controlar el mezclado y la temperatura del líquido contenido dentro del tanque. Además se puede controlar el vaciado (la “purga”) del tanque mediante una válvula.*

*Para modelar la línea de producción en este sistema se definió al proceso productivo como una secuencia de pasos y estos pasos se representaron mediante una jerarquía de clases. A continuación se detalla el esquema para dos de estos pasos:*

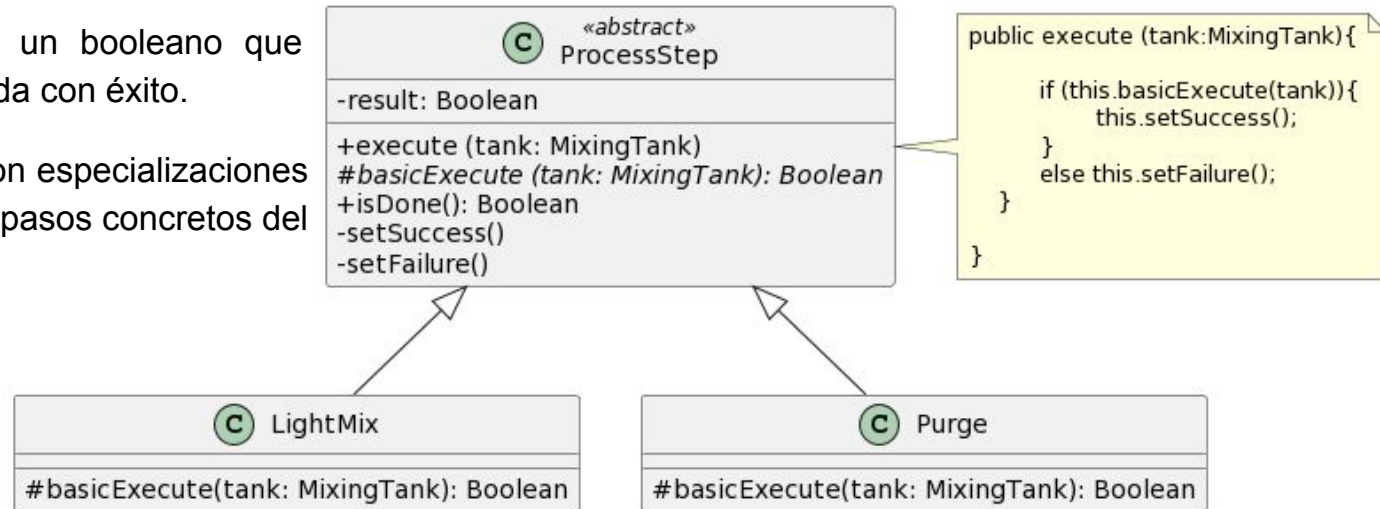
# Ejercicio 22 - Monitoreo de línea de producción

La clase base ProcessStep define la estructura común para todos los pasos.

El método `execute(tank: MixingTank)` recibe como parámetro un tanque mezclador (`MixingTank`) y ejecuta sobre el tanque los comandos de la etapa correspondiente. Para esto invoca al método `basicExecute(tank)` que cada etapa implementa el cual retorna si la ejecución fue o no exitosa;

El método `isDone()` retorna un booleano que describe si la etapa fue realizada con éxito.

Las clases `LightMix` y `Purge` son especializaciones de `ProcessStep` y representan pasos concretos del proceso.



# Ejercicio 22 - Monitoreo de línea de producción

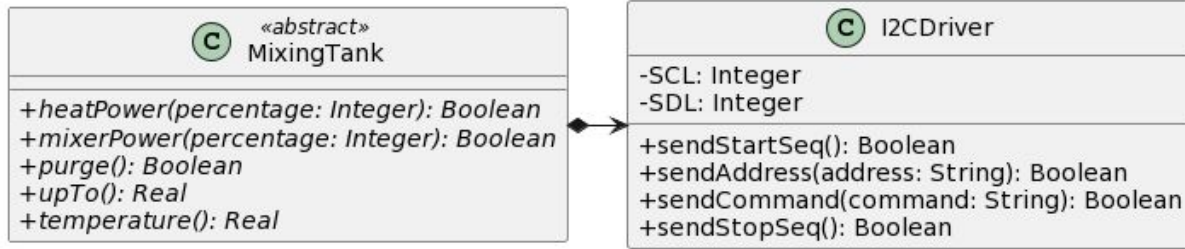
```
clase LightMix
```

```
basicExecute(tank:MixingTank){  
    return tank.heatPower(20%)  
    && tank.mixerPower(5%)  
}
```

```
clase Purge
```

```
basicExecute(tank:MixingTank){  
    return tank.heatPower(0%)  
    && tank.mixerPower(0%)  
    && tank.purge()  
}
```

# Ejercicio 22 - Monitoreo de proceso industrial



No se dispone de una implementación concreta de *MixingTank* pero su comportamiento esperado es el siguiente:

- `heatPower`: configura el nivel potencia de la fuente de calor del tanque de 0 a 100
- `mixerPower`: configura el nivel de potencia de la mezcladora del tanque de 0 a 100
- `purge`: comanda al tanque para que se desagote
- `upTo`: retorna el volumen ocupado del tanque de 0 a 100
- `temperature`: retorna la temperatura del contenido del tanque

# Ejercicio 22 - Monitoreo de proceso industrial

1. Implemente las clases ProcessStep, LightMix y Purge, completando el pseudocódigo provisto.
2. Implemente Test de Unidad para ambas clases cubriendo casos de éxito y falla
  - a. Explique qué tipo de TestDouble es necesario implementar para cubrir esta versión de test cases.
3. Se conocen especificaciones del tanque y se han redefinido las definiciones de LightMix y Purge. Actualice la implementación de clases y de los test cases cubriendo casos de éxito y falla.
  - a. Explique qué tipo de TestDouble es necesario implementar para cubrir esta versión de test cases.

# Ejercicio 22b - Monitoreo de proceso ind. (ext)

## MixingTank

tiempo en completar el método purge: 4 segundos

Transferencia de calor según el nivel de potencia recibido en heatPower (es decir, la velocidad con la que sube la temperatura del tanque mezclador depende de la potencia que se le quiere aplicar):

- 100% = + 5 °C por segundo
- 75% = + 4 °C por segundo
- 50% = + 2 °C por segundo
- 25% = + 1 °C por segundo
- 0% = + 0 °C por segundo

### clase LightMix

```
basicExecute(tank:MixingTank){
    temp = tank.temperature()
    tank.heatPower(100%)
    delay(2sec)
    if(tank.temperature()-temp == 10 ){
        tank.mixerPower(5%)
        return true
    }
    else {
        return false
    }
}
```

### clase Purge

```
basicExecute(tank:MixingTank){
    if (tank.upTo() = 0) {
        return false
    }
    else {
        tank.heatPower(0%)
        tank.mixerPower(0%)
        tank.purge()
        delay(4sec)
        if (tank.upTo() != 0){
            return false
        }
        return true
    }
}
```



# Refactoring - ¿cómo documentar?

- El código pasa los tests -

Algoritmo de refactoring: Realice en forma iterativa los siguientes pasos

1. Indique el mal olor (su nombre y en qué líneas se encuentra)
2. Indique el refactoring que lo/los corrige (el nombre y los pasos involucrados para realizarlo)
3. Aplique el refactoring, mostrando el resultado final (código y/o diseño según corresponda)
4. Si vuelve a encontrar un mal olor, retorne al paso 1.

# Refactoring - Code smell

Windsurf: Refactor | Explain | Generate Javadoc | ✕

```
60 public boolean madreEs(Mamifero otroMamifero) {  
61     return this.madre != null && this.madre == otroMamif  
62 }  
63
```

Windsurf: Refactor | Explain | Generate Javadoc | ✕

```
64 public boolean padreEs(Mamifero otroMamifero) {  
65     return this.padre != null && this.padre == otroMamif  
66 }  
67
```

Windsurf: Refactor | Explain | Generate Javadoc | ✕

```
68 public Mamifero getAbuelaMaterna() {  
69     if (this.madre != null) {  
70         return this.madre.getMadre();  
71     }  
72     return null;  
73 }  
74
```

Windsurf: Refactor | Explain | Generate Javadoc | ✕

```
75 public Mamifero getAbuelaPaterna() {  
76     if (this.padre != null) {  
77         return this.padre.getMadre();  
78     }  
79     return null;  
80 }  
81
```

Windsurf: Refactor | Explain | Generate Javadoc | ✕

Windsurf: Refactor | Explain | Generate Javadoc | ✕

```
82 public Mamifero getAbueloPaterno() {  
83     if (this.padre != null) {  
84         return this.padre.getMadre();  
85     }  
86     return null;  
87 }
```

## Code Smell: Null Check

“Repeating such null logic in one or two places in a system isn’t a problem, but repeating it in multiple places bloats a system with unnecessary code”

## Refactoring: Introduce Null Object

- La lógica para manejarse con un valor nulo en una variable está duplicado por todo el código

# Introduce Null Object - Mecánica

1. Crear el *null object* aplicando “**Extract Subclass**” sobre la clase que se quiere proteger del chequeo por null (clase origen). **Alternativamente** hacer que la nueva clase implemente la misma interface que la clase origen. Compilar.
2. Buscar un null check en el código cliente, es decir, código que invoque un método sobre una instancia de la clase origen si la misma no es null. Redefinir el método en la clase del null object para que implemente el comportamiento alternativo. Compilar
3. Repetir el paso 2 para todos los null checks asociados a la clase origen.
4. Encontrar todos los lugares que pueden retornar null cuando se le pide una instancia de la clase origen. Inicializar con una instancia del null object lo antes posible. Compilar
5. Para cada lugar elegido en el paso 4, eliminar los null checks asociados

# Introduce Null Object - Mecánica

1. Crear el *null object* aplicando “**Extract Subclass**” sobre la clase que se quiere proteger del chequeo por null (clase origen). Alternativamente hacer que la nueva clase implemente la misma interface que la clase origen. Compilar.

```
1 public class NullMamifero extends Mamifero {  
2     public NullMamifero() {  
3         super();  
4     }  
5 }
```

# Introduce Null Object - Mecánica

2. Buscar un null check en el código cliente, es decir, código que invoque un método sobre una instancia de la clase origen si la misma no es null. Redefinir el método en la clase del null object para que implemente el comportamiento alternativo. Compilar

```
1 public boolean padreEs(Mamifero otroMamifero) {  
2     return this.padre != null && this.padre == otroMamifero;  
3 }  
4
```

```
1 @Override  
2 public boolean padreEs(Mamifero otroMamifero) {  
3     return false;  
4 }
```

# Introduce Null Object - Mecánica

2. Buscar un null check en el código cliente, es decir, código que invoque un método sobre una instancia de la clase origen si la misma no es null. Redefinir el método en la clase del null object para que implemente el comportamiento alternativo. Compilar
3. Repetir el paso 2 para todos los null checks asociados a la clase origen.

**Hacemos lo mismo con el resto de los métodos**

# Introduce Null Object - Mecánica

4. Encontrar todos los lugares que pueden retornar null cuando se le pide una instancia de la clase origen. Inicializar con una instancia del null object lo antes posible. Compilar

```
1 public class Mamifero {  
2     private Mamifero padre;  
3  
4     public Mamifero() {  
5         // el constructor inicializa padre en null ...  
6     }  
7 }
```

# Introduce Null Object - Mecánica

4. Encontrar todos los lugares que pueden retornar null cuando se le pide una instancia de la clase origen. Inicializar con una instancia del null object lo antes posible. Compilar

```
1 public class Mamifero {  
2     private Mamifero padre;  
3  
4     public Mamifero() {  
5         this.padre = NullMamifero();  
6     }  
7 }
```



# Introduce Null Object - Mecánica

5. Para cada lugar elegido en el paso 4, eliminar los null checks asociados

```
1 public class Mamifero {
2     public Mamifero getAbuelaPaterna() {
3         if (this.padre ≠ null) {
4             return this.padre.getMadre();
5         }
6         return null;
7     }
8
9     public Mamifero getAbueloPaterno() {
10        if (this.padre ≠ null) {
11            return this.padre.getPadre();
12        }
13        return null;
14    }
15
16    public boolean padreEs(Mamifero otroMamifero) {
17        return this.padre ≠ null && this.padre = otroMamifero;
18    }
19
20    public boolean tieneAncestroPaterno(Mamifero otroMamifero) {
21        return this.padreEs(otroMamifero) || (this.padre ≠ null && this.padre.tieneComoAncestroA(otroMamifero));
22    }
23 }
```

# Introduce Null Object - Mecánica

5. Para cada lugar elegido en el paso 4, eliminar los null checks asociados

```
1 public class Mamifero {
2     public Mamifero getAbuelaPaterna() {
3         if (this.padre != null) {
4             return this.padre.getMadre();
5         }
6         return null;
7     }
8
9     public Mamifero getAbueloPaterno() {
10        if (this.padre != null) {
11            return this.padre.getPadre();
12        }
13        return null;
14    }
15
16    public boolean padreEs(Mamifero otroMamife
17        return this.padre != null && this.padr
18    }
19
20    public boolean tieneAncestroPaterno(Mamife
21        return this.padreEs(otroMamifero) || (
22    }
23 }
```

```
1 public class Mamifero {
2     public Mamifero getAbuelaPaterna() {
3         return this.padre.getMadre();
4     }
5
6     public Mamifero getAbueloPaterno() {
7         return this.padre.getPadre();
8     }
9
10    public boolean padreEs(Mamifero otroMamifero) {
11        return this.padre == otroMamifero;
12    }
13
14    public boolean tieneAncestroPaterno(Mamifero otroM
15        return this.padreEs(otroMamifero) || (this.pac
16    }
17 }
```

# Frameworks - SingleThreadTCPFramework

- Leer el material
- En los documentos se explican conceptos teóricos, cómo funciona y cómo se usa  
SingleThreadTCPFramework



Framework: SingleThreadTCPServer



Presentación: Reuso, Librerías y Frameworks



OO2\_SingleThreadTCPServer.pdf



SingleThreadTCPServer.zip

## Para ejecutar el código, en una terminal / consola:

1. Compilar:

```
javac EchoServer.java VoidServer.java SingleThreadTCPServer.java
```

2. Ejecutar la **clase concreta** que queremos instanciar, indicando también un **número de puerto**:

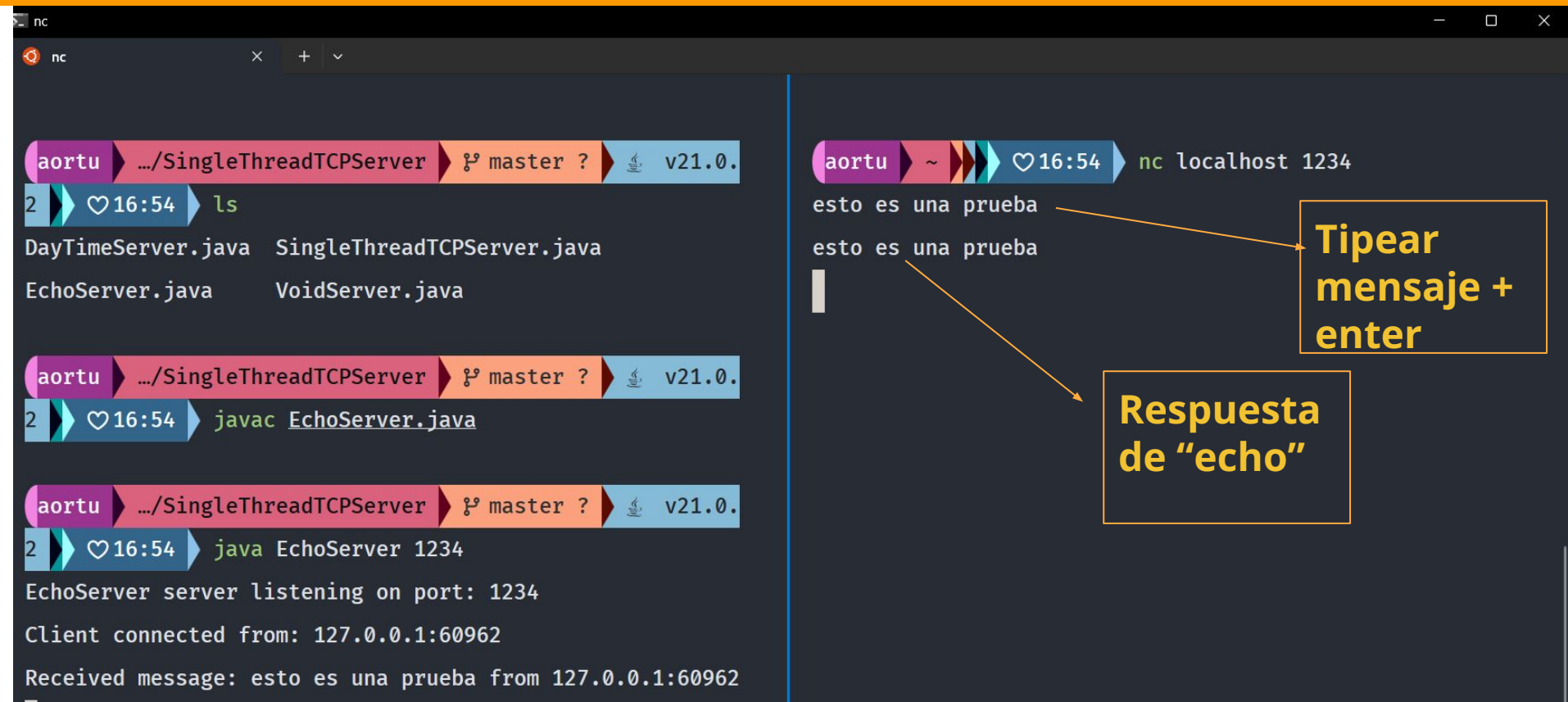
```
java VoidServer 8888
```

```
java EchoServer 1234
```

3. **Conectarse con algún cliente para enviar un mensaje y recibir la respuesta**

- **Linux:** se puede usar Netcat (commando **nc**)
- **Windows:**
  - **Windows Subsystem for Linux (WSL)**, usar jdk y luego usar Netcat
  - **Telnet:** se usa por consola de manera similar. Requiere habilitarlo en "características de Windows"
  - **PuTTY:** Es otro cliente Telnet

# Ejecución - Linux / Windows con WSL



```
nc
aortu .../SingleThreadTCPServer ? master ? v21.0.
2 ? 16:54 ls
DayTimeServer.java SingleThreadTCPServer.java
EchoServer.java VoidServer.java

aortu .../SingleThreadTCPServer ? master ? v21.0.
2 ? 16:54 javac EchoServer.java

aortu .../SingleThreadTCPServer ? master ? v21.0.
2 ? 16:54 java EchoServer 1234
EchoServer server listening on port: 1234
Client connected from: 127.0.0.1:60962
Received message: esto es una prueba from 127.0.0.1:60962
```

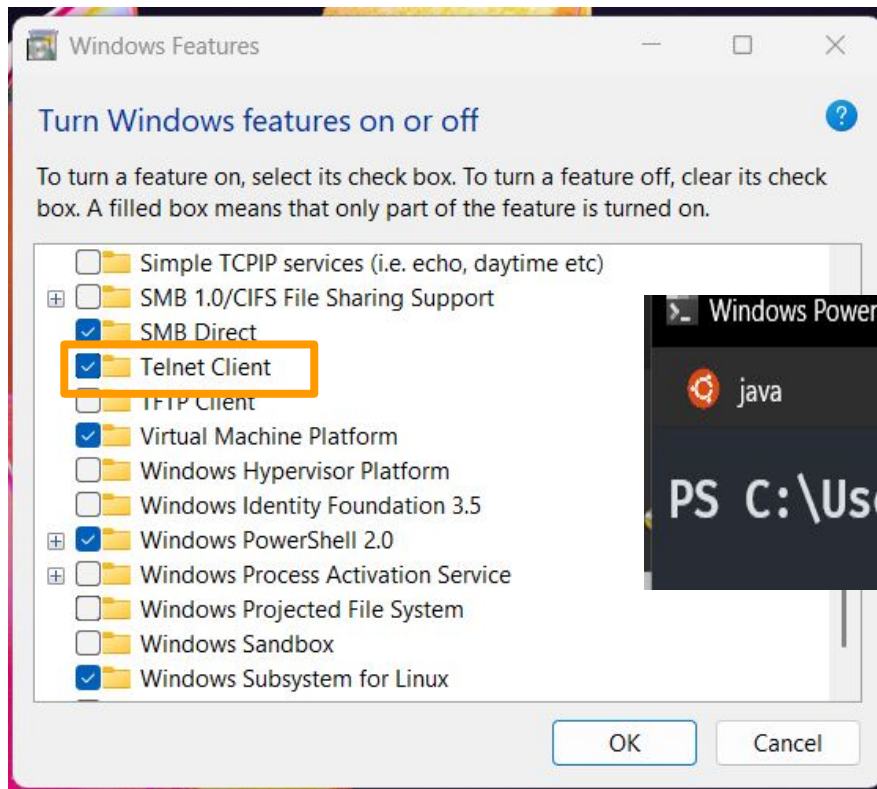
esto es una prueba

esto es una prueba

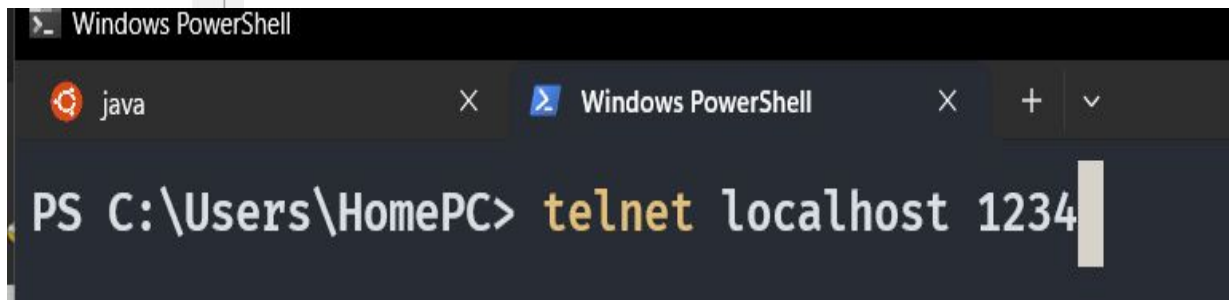
**Tipear mensaje + enter**

**Respuesta de "echo"**

# Ejecución - Windows (Telnet)



Panel de control > Programas >  
Activar o desactivar características  
de Windows

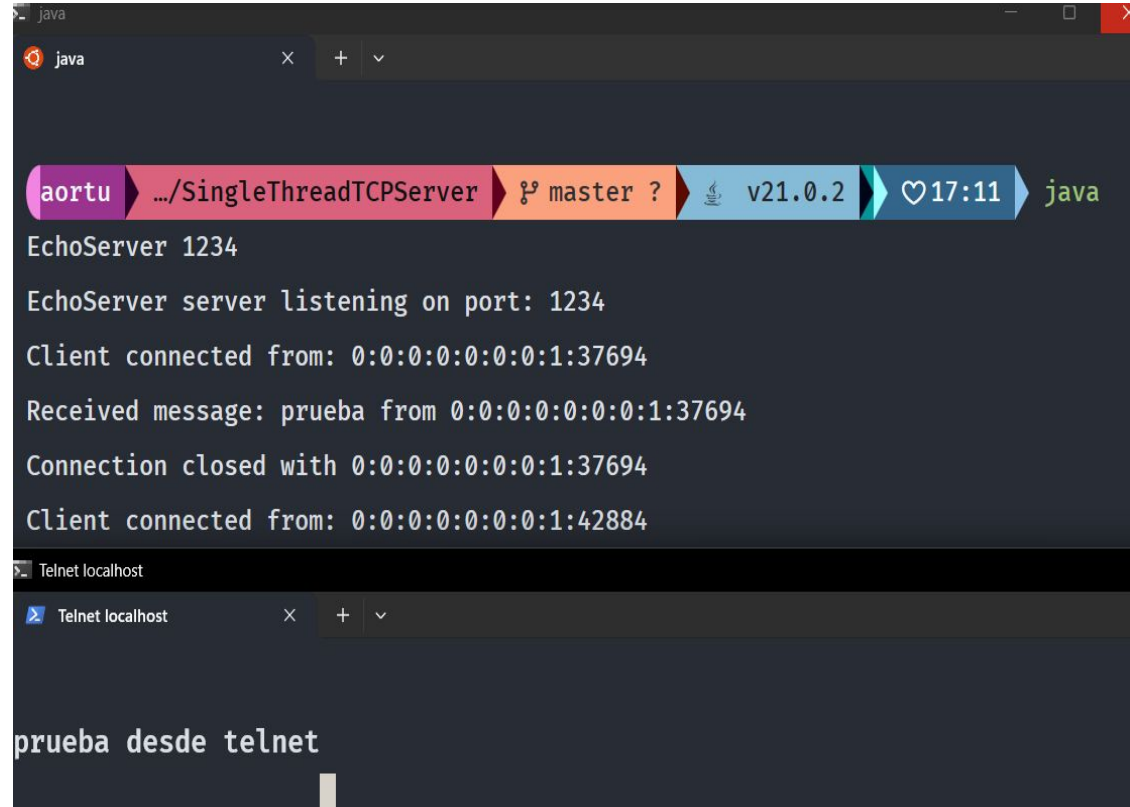


# Ejecución - Windows (Telnet)

Abrir un Powershell y escribir

```
telnet localhost puerto
```

Luego tipear mensaje + enter



The screenshot shows two windows. The top window is a Java IDE with a dark theme. It has a breadcrumb trail at the top: 'aortu' (purple), '.../SingleThreadTCPServer' (pink), 'master ?' (orange), 'v21.0.2' (light blue), and '17:11' (dark blue). The main text area shows the following output:

```
EchoServer 1234
EchoServer server listening on port: 1234
Client connected from: 0:0:0:0:0:0:0:1:37694
Received message: prueba from 0:0:0:0:0:0:0:1:37694
Connection closed with 0:0:0:0:0:0:0:1:37694
Client connected from: 0:0:0:0:0:0:0:1:42884
```

The bottom window is titled 'Telnet localhost' and has a single tab labeled 'Telnet localhost'. It shows the text 'prueba desde telnet' at the bottom, with a white cursor line positioned below it.

# Frameworks - Ejercicio 1

Refactorizar el método

**SingleThreadTCPFramework::handleClient(Socket)** para convertirlo en un Template Method.

- Este método debe incluir métodos hook opcionales, es decir, métodos hooks que pueden ser implementados por las subclases o no.
- ¿Qué cosas creemos que podrían variar cuando un programador utiliza este framework y quiere construir una aplicación?



```
1 private final void handleClient(Socket clientSocket) {
2     try (
3         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
4         BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));) {
5         String inputLine;
6         while ((inputLine = in.readLine()) != null) {
7             System.out.println("Received message: " + inputLine + " from "
8                 + clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort());
9
10            if ("".equalsIgnoreCase(inputLine)) {
11                break; // Client requested to close the connection
12            }
13            this.handleMessage(inputLine, out);
14        }
15        System.out.println("Connection closed with " + clientSocket.getInetAddress().getHostAddress() + ":"
16            + clientSocket.getPort());
17    } catch (IOException e) {
18        System.err.println("Problem with communication with client: " + e.getMessage());
19    } finally {
20        try {
21            clientSocket.close();
22        } catch (IOException e) {
23            System.err.println("Error closing socket: " + e.getMessage());
24        }
25    }
26 }
```

```
1 private final void handleClient(Socket clientSocket) {
2     try (
3         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
4         BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));) {
5         String inputLine;
6         while ((inputLine = in.readLine()) != null) {
7             System.out.println("Received message: " + inputLine + " from "
8                 + clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort());
9
10            if ("".equalsIgnoreCase(inputLine)) {
11                break; // Client requested to close the connection
12            }
13            this.handleMessage(inputLine, out);
14        }
15        System.out.println("Connection closed with " + clientSocket.getInetAddress().getHostAddress() + ":"
16            + clientSocket.getPort());
17    } catch (IOException e) {
18        System.err.println("Problem with communication with client: " + e.getMessage());
19    } finally {
20        try {
21            clientSocket.close();
22        } catch (IOException e) {
23            System.err.println("Error closing socket: " + e.getMessage());
24        }
25    }
26 }
```

```
1 private final void handleClient(Socket clientSocket) {
2     try (
3         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
4         BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));) {
5         String inputLine;
6         while ((inputLine = in.readLine()) != null) {
7             this.beforeCommunication(clientSocket, inputLine);
8
9             if ("".equalsIgnoreCase(inputLine)) {
10                 break; // Client requested to close the connection
11             }
12             this.handleMessage(inputLine, out);
13         }
14         System.out.println("Connection closed with " + clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort());
15     }
```

```
1 protected void beforeCommunication(Socket clientSocket, String inputLine) {
2     System.out.println("Received message: " +
3         inputLine + " from " +
4         clientSocket.getInetAddress().getHostAddress() +
5         ":" + clientSocket.getPort());
6 }
```

```
24 }
25 }
```

```
1 protected void beforeCommunication(Socket clientSocket, String inputLine) {  
2     System.out.println("Received message: " +  
3         inputLine + " from " +  
4         clientSocket.getInetAddress().getHostAddress() +  
5         ":" + clientSocket.getPort());  
6 }
```

¿Que podría hacer un programador al instanciar el framework y redefinir éste método? Sólo algunas ideas...

- **Auditoría:** guardar en un log / base de datos quién accedió, desde que IP, en qué fecha/hora ...
- **Seguridad:** implementar alguna política para que ciertos accesos no sean permitidos

```

1 private final void handleClient(Socket clientSocket) {
2     try (
3         PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
4         BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));) {
5         String inputLine;
6         while ((inputLine = in.readLine()) != null) {
7             this.beforeCommunication(clientSocket, inputLine);
8
9             if ("".equalsIgnoreCase(inputLine)) {
10                 break; // Client requested to close the connection
11             }
12             this.handleMessage(inputLine, out);
13         }
14         System.out.println("Connection closed with " + clientSocket.getInetAddress()
15             + clientSocket.getPort());
16     } catch (IOException e) {
17         System.err.println("Problem with communication with client: " + e.getMessage());
18     } finally {
19         try {
20             clientSocket.close();
21         } catch (IOException e) {
22             System.err.println("Error closing socket: " + e.getMessage());
23         }
24     }
25 }

```

- ¿Qué otras cosas creemos que podrían variar cuando un programador utiliza este framework y quiere construir una aplicación?



