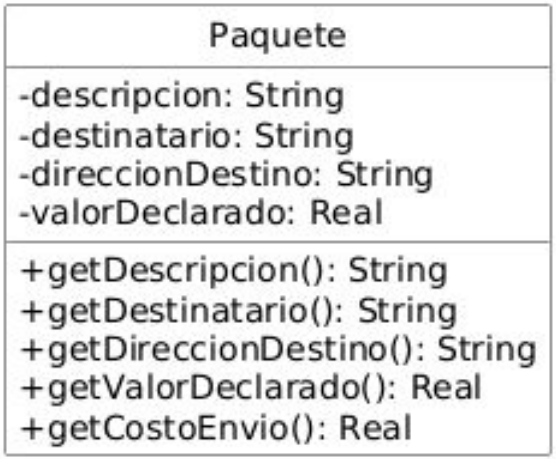
Orientación a Objetos II 2025

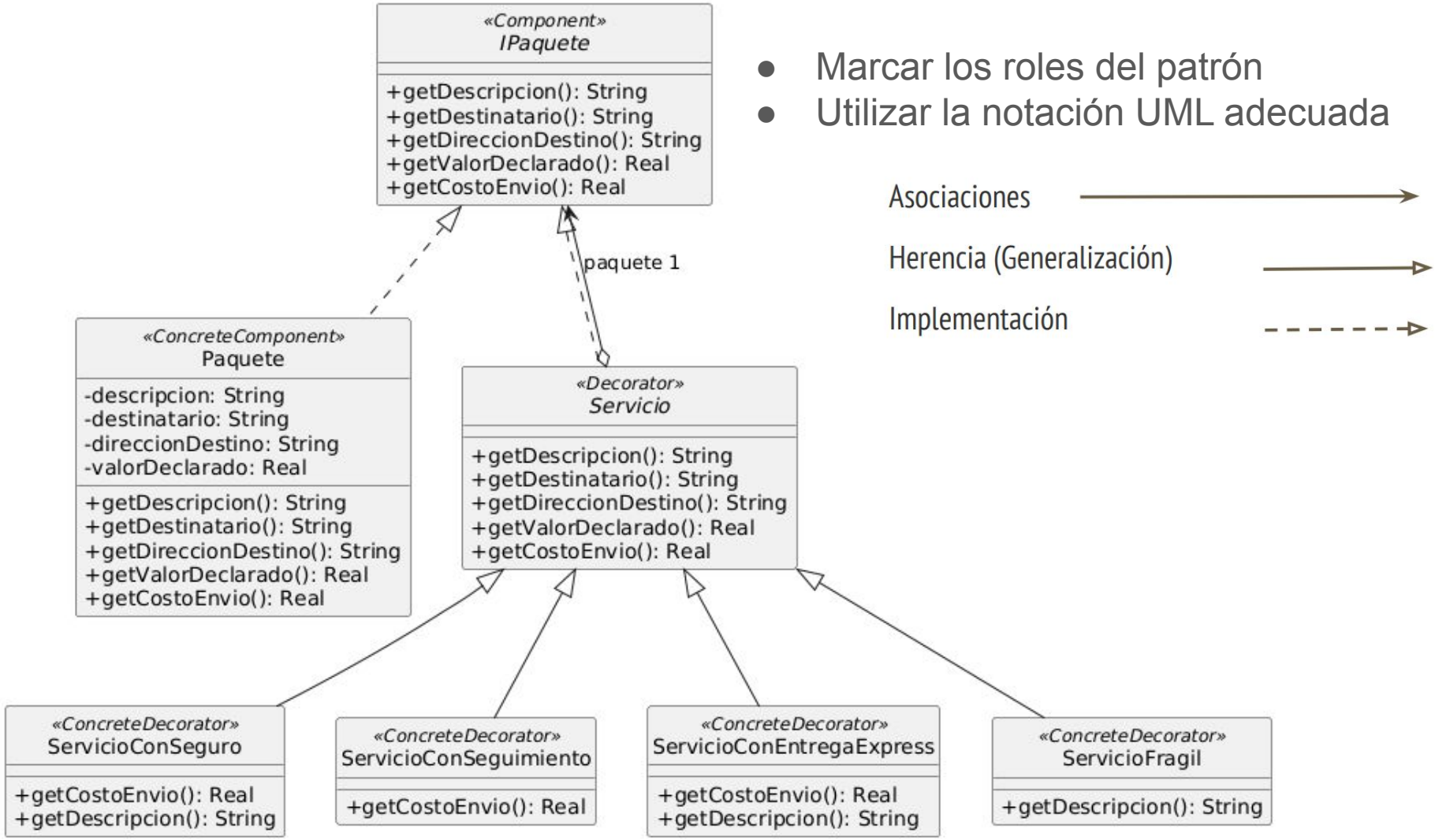
Explicación 1era fecha

Ejercicio 1 - Patrones



Se plantea la incorporación de servicios adicionales [...], sin modificar la clase Paquete, pero permitiendo que la clase pueda implementar una interfaz en caso de ser necesario.
Se pide que los servicios puedan combinarse libremente.

- **Con seguro**: Incrementa el costo de envío en un 20% del valor declarado y agrega la nota "con seguro" a la descripción.
- **Con seguimiento en tiempo real**: Incrementa el costo de envío en 2000 pesos, pero no modifica la descripción.
- **Entrega exprés**: Incrementa el costo de envío en un 50% del valor declarado y agrega la nota "entrega express" a la descripción.
- **Manipulación frágil**: No incrementa el costo, pero agrega la nota "frágil" a la descripción.

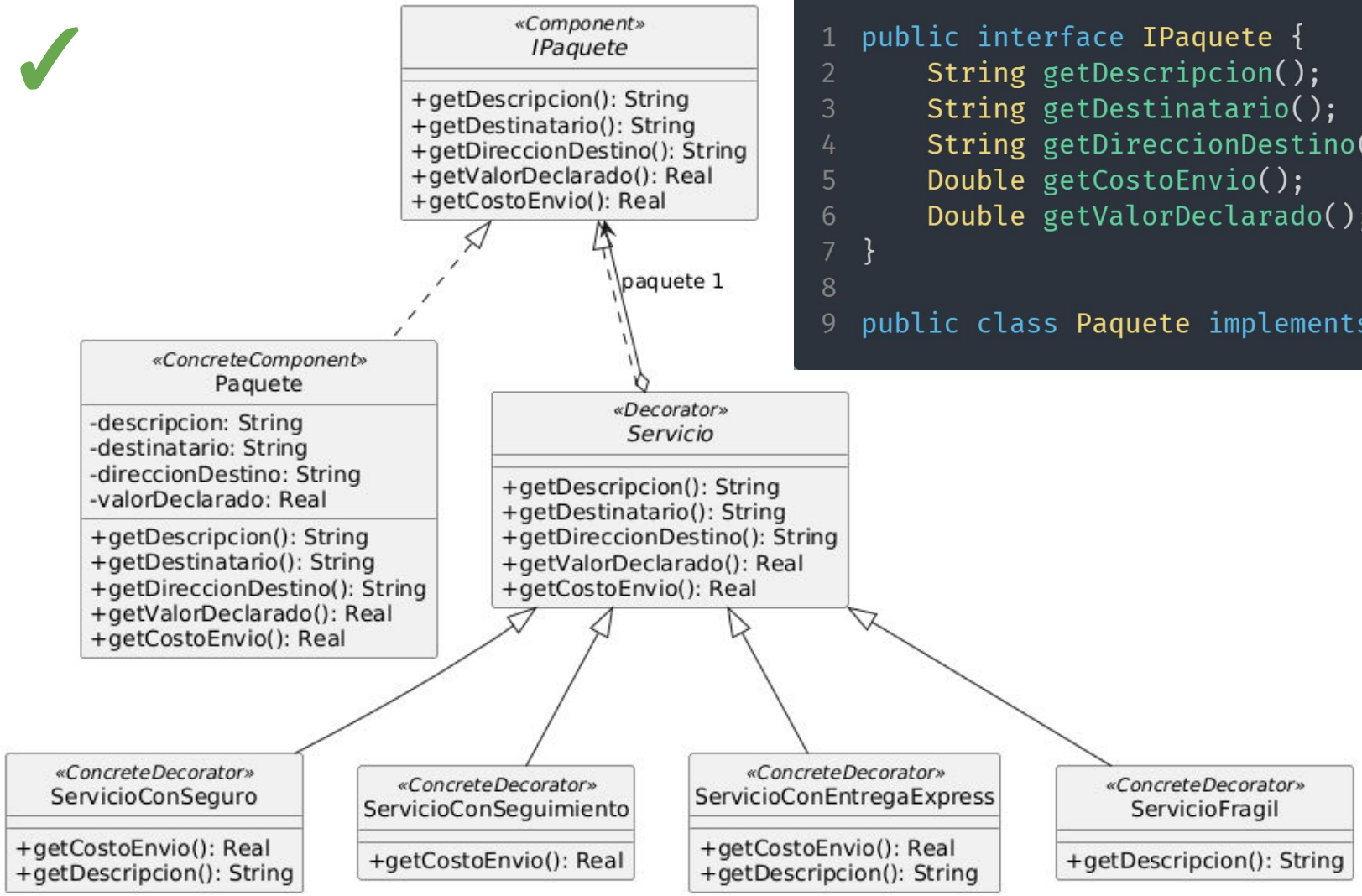


- Marcar los roles del patrón
- Utilizar la notación UML adecuada

Asociaciones

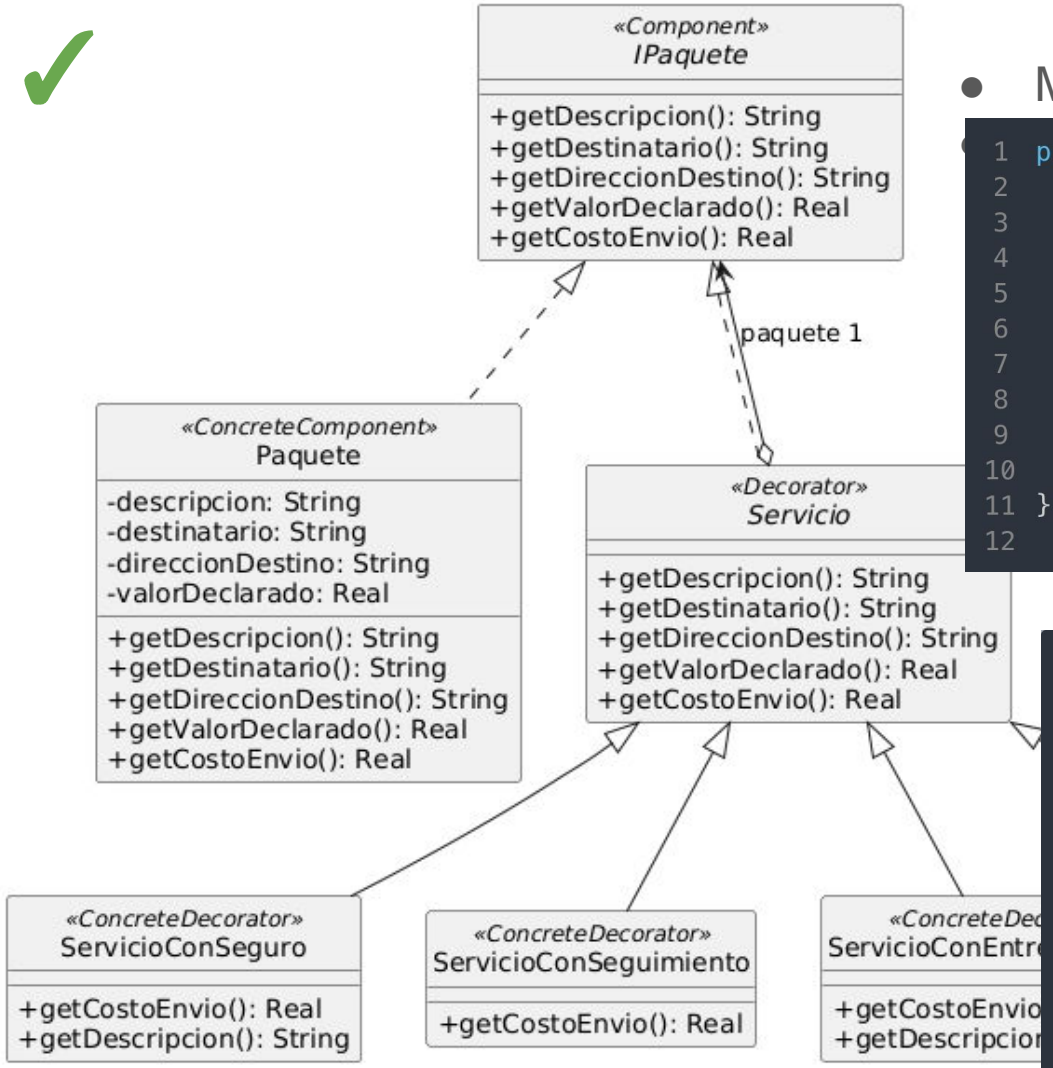
Herencia (Generalización)

Implementación




```java
public interface IPaquete {
    String getDescripcion();
    String getDestinatario();
    String getDireccionDestino();
    Double getCostoEnvio();
    Double getValorDeclarado();
}

public class Paquete implements IPaquete {}
```



- Marcar los roles del patrón

```java
public abstract class Servicio implements IPaquete {
    private IPaquete paquete;

    public Servicio(IPaquete paquete) {
        this.paquete = paquete;
    }

    public String getDescripcion() {
        return this.paquete.getDescripcion();
    }
}
```

```java
public class ServicioConSeguro extends Servicio {
    public ServicioConSeguro(IPaquete paquete) {
        super(paquete);
    }

    @Override
    public String getDescripcion() {
        return super.getDescripcion() + " con seguro";
    }

    @Override
    public Double getCostoEnvio() {
        return super.getCostoEnvio() +
                (this.getValorDeclarado() * 0.2);
```

Ejercicio 2 - Refactoring

Tareas (debe realizar los tres ítems para aprobar el tema)**:**

1. Enumere los code smell que encuentra en el código indicando las líneas afectadas.
2. Indique que refactorings utilizará para solucionarlos considerando que se desea incluir nuevos formatos para exportar (XLSX, CSV entre otros). Explique los pasos necesarios para realizar los refactorings elegidos, haciendo referencia al código cuando corresponda. Muestre el código final resultado luego de aplicar esos refactorings en la clase ReportGenerator y si hace falta, en la clase ReportGeneratorTest.
3. Realice el diagrama de clases del código refactorizado. Si utilizó un patrón de diseño, indíquelo en el diagrama mostrando los roles del patrón.

Ejercicio 2 - Refactoring

Enumere los code smell que encuentra en el código indicando las líneas afectadas.

1. Long method lineas 5-44
2. Switch statement lineas 6 y 25
3. Duplicated code lineas 8-13 y 27-32
4. ….

No es necesario explicar por qué son malos olores, con enumerarlos es suficiente

Esta es una posible solución…

Ejercicio 2 - Refactoring

2. Indique que refactorings utilizará para solucionarlos [...]. **Explique los pasos necesarios para realizar los refactorings elegidos, haciendo referencia al código cuando corresponda**

Para el código repetido, aplicamos extract method en las líneas 8-13. Pasos:

1. Creamos un nuevo método privado con nombre `crearDocumento(Document document)`
2. Copiamos el código de las líneas 8-13 al método `crearDocumento`
3. Reemplazamos las líneas 8-13 con un llamado a `crearDocumento`
4. Reemplazamos las líneas 27-32 con un llamado a `crearDocumento`

Ejercicio 2 - Refactoring

Por ejemplo, una forma de resolverlo:

Para el switch statement, aplicamos aplicamos “Replace conditional with polymorphism”. Pasos:

1. Creamos dos subclases de ReportGenerator: PdfReportGenerator y XlsReportGenerator
2. En cada subclase, creamos el mensaje `generateReport` y copiamos las líneas de la rama del condicional correspondiente. Líneas 7(*)-24 y 26(*)-43
3. Luego de copiar el código, borramos la condición y el código de la rama del condicional en la superclase
4. Hacemos que el método `generateReport` de la clase ReportGenerator sea abstracto
5. Adecuar el constructor y su invocación desde el test
6. Eliminamos la variable de instancia tipo

Ejercicio 2 - Refactoring

Bad smell luego de crear la jerarquía: Las subclases realizan pasos similares en el mismo orden en el método `generateReport`, pero los pasos son distintos. Se refactoriza aplicando **Form Template Method.**

Pasos:

1. Extraer las partes de `generateReport` en métodos idénticos (misma firma y cuerpo en las subclases) o métodos unicos.
   a. Se crean los métodos: **indicando los nombres y líneas que contienen**
2. Aplicar **Pull Up Method** para los métodos idénticos: **indicando los nombres**
3. Aplicar **Rename Method** para los métodos únicos hasta que el método similar quede con cuerpo idéntico en las subclases: **indicando los nombres**
4. Aplicar **Rename Method** sobre los métodos similares de las subclases: **indicando los nombres**
5. Aplicar **Pull Up Method** sobre los métodos idénticos: **indicando los nombres**
6. Definir métodos abstractos en la superclase por cada método único de las

Ejercicio 2 - Refactoring

Bad smell luego de crear la jerarquía: Las subclases realizan pasos similares en el mismo orden en el método `generateReport`, pero los pasos son distintos. Se refactoriza aplicando **Form Template Method.**

Pasos:

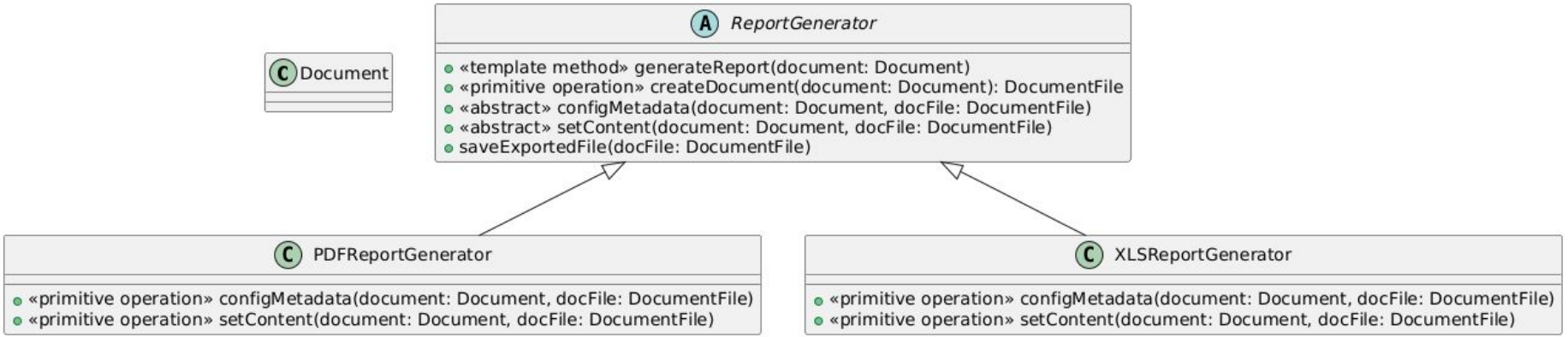
1. Extraer las partes de `generat` `ort` en métodos idénticos (misma firma y cuerpo en las subclases) o mét `s`.
   a. Se crean los métodos: **indic** **bres y líneas que contienen**
2. Aplicar **Pull Up Metho**
3. Aplicar **Rename Meth** quede con cuerpo idén
4. Aplicar **Rename Method** sobre los métodos similares de las subclases: **indicando los nombres**
5. Aplicar **Pull Up Method** sobre los métodos idénticos: **indicando los nombres**
6. Definir métodos abstractos en la superclase por cada método único de las subclases : **indicando los nombres**

Se debe indicar en el diagrama UML los roles del template method resultante

```java
public class ReportGenerator {

    public void generateReport(Document document) {
        DocumentFile docFile = this.createDocument(document);
        this.configMetadata(document, docFile);
        this.setContent(document, docFile);
        this.saveExportedFile(docFile);
    }

    ...
```



Ejercicio 2 - Errores más comunes

En este ejercicio se evalúa el **proceso de refactoring,** además de la codificación (que debe corresponder con el proceso)

1. Falta detallar los pasos de los refactoring realizados
   a. Se deben listar los pasos del refactoring utilizado (si es necesario), indicando en cada paso, cómo se modifica el código (nombre del mensaje, invocación, etc).
2. El código final entregado no coincide con los pasos de refactoring aplicados
   a. Debe ser posible aplicar los pasos de los refactorings utilizados y llegar al código y diseño final
3. Resolver varios refactoring en un sólo paso. Cada refactoring debería ser atómico y deberíamos tener un código funcional cuando termina
4. Errores de mecánica del refactoring
   a. No se respetan los pasos del refactoring elegido
5. Código final con errores (ya no funciona el código, tiene errores importantes)
6. Código final que cambia el comportamiento del programa

Ejercicio 3 - Frameworks

Tareas (debe realizar los cuatro ítems para aprobar el tema)**:**

1. Explique brevemente cuáles son los frozen spot y hot spot del código presentado del framework.
2. Modificar este framework para permitir personalizar los mensajes que se muestran en consola por defecto cuando un cliente se conecta y se desconecta
3. Explique brevemente cuáles son los frozen spot y hot spot después de la extensión realizada.
4. Indique en dónde se produce la inversión de control en la extensión realizada.

Ejercicio 3 - Frameworks

```java
private final void handleClient(Socket clientSocket) {
    try {
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        BufferedReader in = new BufferedReader(
                                        new  InputStreamReader(clientSocket.getInputStream()));
        String inputLine;
        while ((line = in.readLine()) != null) {
            System.out.println("Received message: " + inputLine
                    + " from "  + clientSocket.getInetAddress().getHostAddress()
                    + ":" + clientSocket.getPort());

            if (line.isEmpty()) break;
            this.handleMessage(line, out);
        }
        System.out.println("Connection closed with " + clientSocket.getInetAddress().getHostAddress()
                + ":" + clientSocket.getPort());

    } catch (IOException e) {
        System.err.println("Error: " + e.getMessage());
    } finally {
        try {
            clientSocket.close();
        } catch (IOException ignored) {}
    }
}
```