GUAYMAS MATIAS legajo: 2306110 — HOJA 1/6 — 07 06 25

Refactoring

① Se actualizan las referencias. Ahora en las líneas 8-16, 19-21, ~~24~~, 27-35, 38-40, se llaman a los métodos extraídos

1.2. Bad Smell: Long Method (línea 5 a 44) ✓

Refactoring: Extract Method

① {
- Se extrae la funcionalidad de crear Documento y configurar metadatos → Método configMetadatos ✓
- ~~Se extrae la funcionalidad de crear Exportador y setear el contenido~~ → método crear Export para PDF Excel
- ~~Se extrae la funcionalidad de ...~~

Bad Smell: Comments (línea 7, 18, 23, 26, 37, 42) ✓ — Bueno A

Refactoring: Eliminación de comentarios (debe ser autoexplicativo). ✓ — Report. A

Bad Smell: Switch Statement (línea 6, 25) — framework A

Refactoring: Replace Logic Conditional with Strategy. ✓

1) Crear clase Strategy ✓ — Germán

2) Move Method de los 4 métodos extraídos anteriormente. ✓ — Visado

1. Modificar la v.i. type para que ahora referencie a la clase Strategy
2. Dejar un método (generateReport (...)) que delegue a la estrategia el comportamiento asociado en clase Report Generator
3. ~~Proveer un setStrategy que reciba una estrategia por parámetro y la setea~~
4. ~~Proveer a la clase Strategy lo necesario para su funcionamiento~~

✗ 3) Extract Parameter para que en el Constructor se reciba como parámetro Strategy y sea setteada

4) Replace Conditional with Polymorphism + Form Template Method: hay código duplicado y se podría definir un método plantilla que invoque pasos en común y que en este caso dos de ellos sean abstractos (la funcionalidad de crear Exportador y settear contenido). ✓

Se crearán nuevas estrategias concretas encargadas de implementar esos métodos / Form template para líneas 15-16 y 34-35 / 19-21 y 38-40.

(*)

```
public class ReportGenerator {
    private Strategy strategy;
    public ReportGenerator(Strategy strategy){ this.strategy = strategy; }
    public void generateReport (Document document){
Document file doc = this.strategy.generateReport (document);
        this.saveExportToFile (doc);
    }
    public void setStrategy (Strategy strategy){
        this.strategy = strategy;
    }
```

(*) Bad Smell: ~~Reinventando la rueda (líneas 11-13, 30-32)~~ ✓

Refactoring: Replace Loop with Pipeline (Streams)

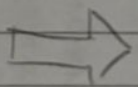
```
public abstract class Strategy {
    public DocumentFile generateReport(Document document) {
        DocumentFile docFile = this.configMetadatos(document);
        PDFExporter = this.crearExporter(document, docFile);
        return docFile;
    }
    private DocumentFile configMetaDatos(Document) document) {
        DocumentFile docFile = new DocumentFile();
        docFile.setTitle(Document.getTitle());
        docFile.setAuthor(this.Document.getAuthors().stream().collect(Collectors.joining(a -> author)))
        this.setAO(document, docFile);
        return docFile;
    }
    protected abstract void crearExporter(Document doc, DocumentFile docFile);
    protected void setAO(Document doc, DocumentFile docFile);
}
public abstract void
public class PDFStrategy extends Strategy {
    protected void crearExporter(Document doc, DocumentFile docFile){
        PDFExporter exporter = new PDFExporter();
        byte[] content = exporter.generatePDFfile(doc);
        docFile.setContent(content);
    }
    protected void setAO(Document doc, DocumentFile docFile){
        docFile.contentType("application/pdf");
        docFile.setPageSize("A4");
    }
}
public class XLSStrategy extends Strategy {
    protected void crearExporter(Document doc, DocumentFile docFile){
        ExcelWriter writer = new ExcelWriter();
        byte[] content = writer.generateExcelFile(doc);
        docFile.setContent(content);
    }
    protected void setAO(Document doc, DocumentFile docFile){
        docFile.contentType("application/vnd.ms-excel");
        docFile.setSheetName(doc.getSubtitle());
    }
}
```

Se debe llamar desde el template

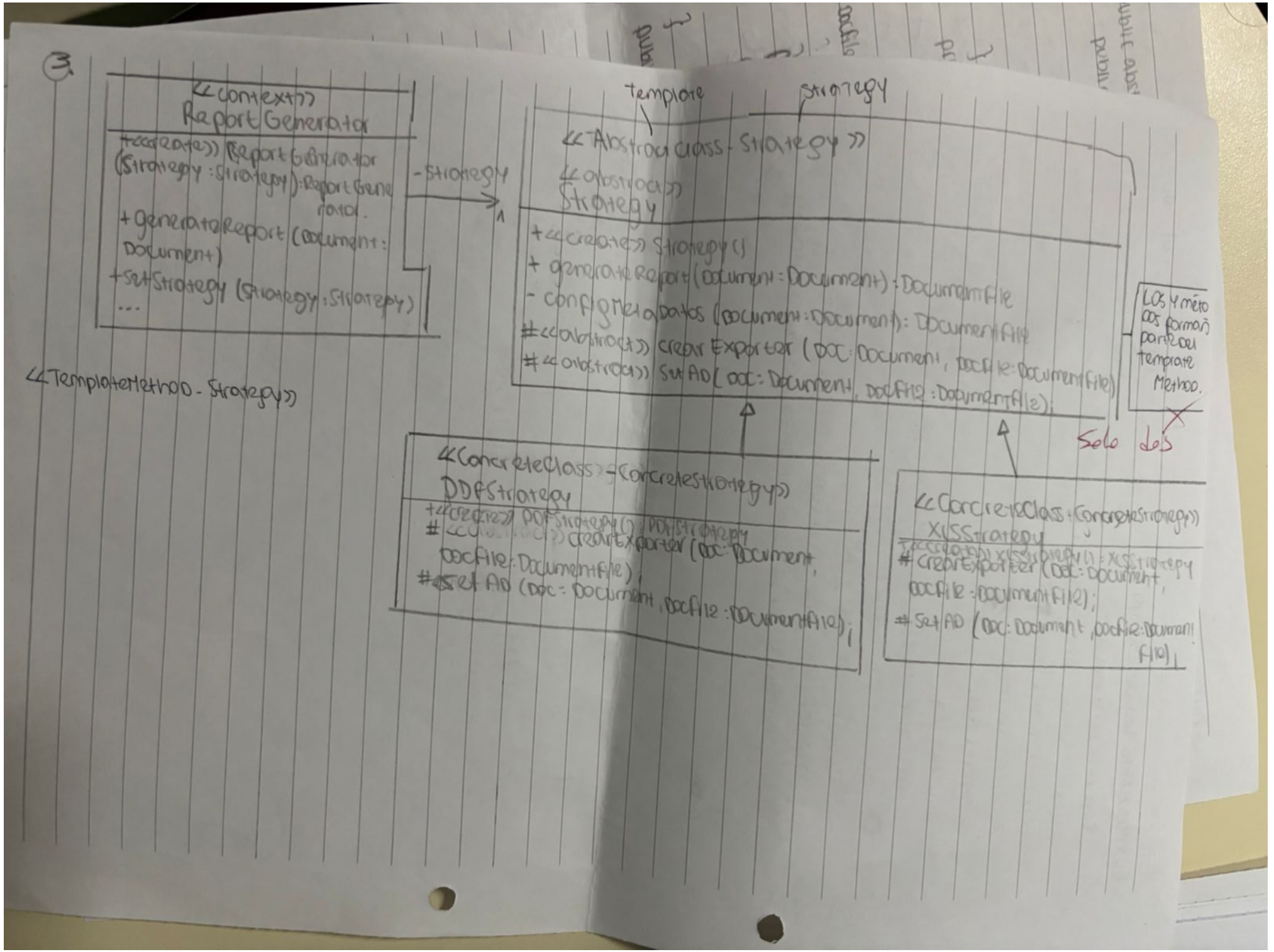
extraer a un método

GUAYMAS MATIAS Legajo: 23061/0 HOJA 2/6 07 06 25

```
class ReportGeneratorTest {
    ReportGenerator generatorPDF;
    ReportGenerator generatorXLS;
    Document document;
    @BeforeEach
    void setUp() {
        document = new Document("Informe");
        document.addAuthor("Carlos");
        document.addAutor("Ana");
        generatorPDF = new ReportGenerator(new PDFStrategy());
        generatorXLS = new ReportGenerator(new XLSStrategy());
    }
    @Test
    void testPDF() {
        generatorPDF.generateReport(document);
        //...
    }
    @Test
    void testXLS() {
        generatorXLS.generateReport(document);
        //...
    }
```



3.


<<Context>>
ReportGenerator
+<<create>> ReportGenerator (Strategy:Strategy):ReportGenerator.
+generateReport(Document: Document)
+setStrategy(Strategy:Strategy)
...
-Strategy
template
strategy
<<Abstract class - Strategy>>
<<abstract>>
Strategy
+<<create>> Strategy()
+generateReport(Document:Document):DocumentFile
-configMetaDatos(Document:Document):DocumentFile
#<<abstract>> crearExporter(doc:Document, docFile:DocumentFile)
#<<abstract>> setAD(doc:Document, docFile:DocumentFile);
Los 4 métodos forman parte del Template Method.
Solo
dos
<<TemplateMethod - Strategy>>
<<ConcreteClass>> - <<ConcreteStrategy>>
PDFStrategy
+<<create>> PDFStrategy():PDFStrategy
#<<override>> crearExporter(doc:Document, docFile:DocumentFile);
#setAD(doc:Document, docFile:DocumentFile);
<<ConcreteClass - ConcreteStrategy>>
XLSStrategy
+<<create>> XLSStrategy():XLSStrategy
#crearExporter(doc:Document, docFile:DocumentFile);
#setAD(doc:Document, docFile:DocumentFile);

GUAYMAS MATIAS Legajo: 230610 HOJA 3/6 07 06 25

Frameworks

1. Los frozenspot del código es decir el comportamiento invariable del framework, se corresponde a la lógica desde la línea 1 hasta la 12, y de la línea 14 hasta la 25. Toda esta lógica ya se encuentra implementada y el programador no puede modificarla, pero existe un hotspot y se encuentra en la línea 13, el mensaje handleMessage(line, out) permite al programador cambiar el comportamiento extendiendo la clase SingleThreadTCPServer y definiendo ese método.

2. Para personalizar este framework y permitir la personalización de mensajes que se muestran en consola por defecto cuando:

- un cliente se conecta: en la línea 11 agregaría un hook method con nombre displayConnection Message() que sea abstracto y que todas las clases que extiendan a SingleThreadTCPServer deban implementar la personalización de este mensaje de bienvenida. Se podría mantener o eliminar el display de las líneas 8 a 10 según la preferencia del programador.
- un cliente se desconecta: en la línea 17 agregaría otro hook method con nombre displayExitMessage() que sea abstracto y que todas las clases que extiendan a SingleThreadTCPServer deban implementar la personalización de este mensaje de desconexión. Se podría mantener o eliminar el display de las líneas 15 a 16 según la preferencia del programador.

Quedarían entonces:

Línea 11. this.displayConnectionMessage();
Línea 17. this.displayExitMessage();

Falta pasar la instancia de Socket

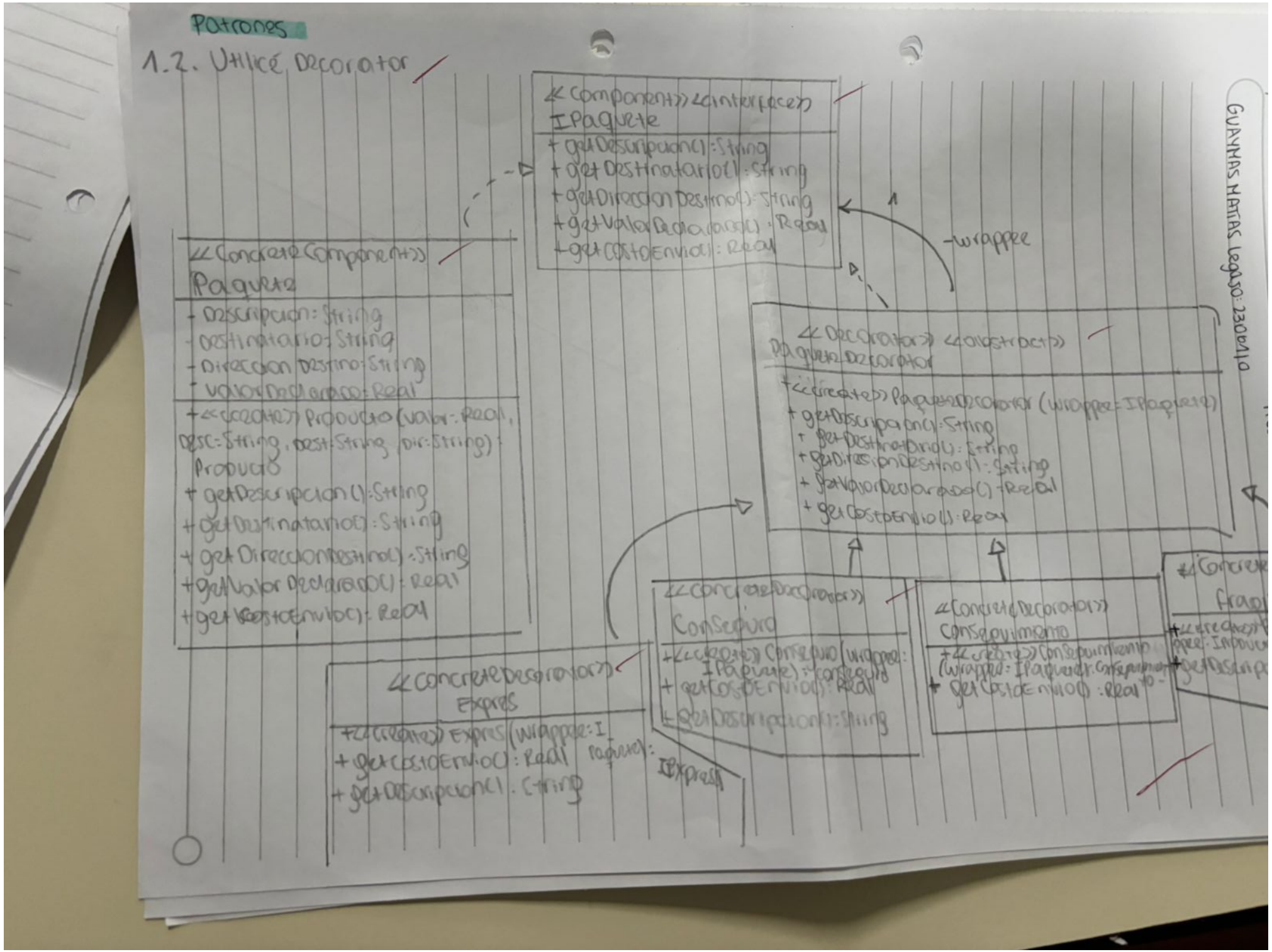
public abstract void displayConnectionMessage();
public abstract void displayExitMessage();

3. Los frozen spot serían toda la lógica asociada a las líneas 1 a 10, 12 a 16 y 18 a 25. Y los hotspots nuevos serían los agregados en la línea 11 y 17 que funcionan como ganchos, además de la línea 13 anteriormente mencionada.

4. En la extensión realizada el framework toma el control (inversión de control) en las líneas 11 y 17. Si no hablamos de la extensión, la línea 13 se considera como otro punto de inversión de control

Patrones

1.2. Utilicé Decorator



GUAYMAS MATIAS Legajo: 230041/0

GUAYMAS MATIAS Legajo 230611/0 HOJA 5/6 07 06 25

3.
```
public interface IPaquete{
    public String getDescripcion();
    public String getDestinatario();
    public String getDireccionDestino();
    public Double getValorDeclarado();
    public double getCostoEnvio();
}
public abstract class PaqueteDecorator implements IPaquete{
    private IPaquete wrappee;
    public PaqueteDecorator(IPaquete wrappee){
        this.wrappee=wrappee;
    }
    public String getDescripcion(){
        return this.wrappee.getDescripcion();
    }
    public String getDestinatario(){
        return this.wrappee.getDestinatario();
    }
    public String getDireccionDestino(){
        return this.wrappee.getDireccionDestino();
    }
    public Double getValorDeclarado(){
        return this.wrappee.getValorDeclarado();
    }
    public double getCostoEnvio(){
        return this.wrappee.getCostoEnvio();
    }
}
```

```
public Conseguro (IPaquete wrappee){
        super(wrappee)
    }
public class Conseguro extends PaqueteDecorator{
    @Override double
    public getCostoEnvio(){
        return super.getCostoEnvio() + this.getValorDeclarado() * 1.2;
    }
    @Override
    public String getDescripcion(){
        return super.getDescripcion() + "con seguro";
    }
public class Conseguimiento extends PaqueteDecorator{
    @Override
    public double getCostoEnvio(){
        return super.getCostoEnvio() + 2000;
    }
}
public Conseguimiento(IPaquete wrappee){
    super(wrappee)
}
public class Expres extends PaqueteDecorator{
    public Expres (IPaquete wrappee){
        super(wrappee);
    }
    @Override
    public double getCostoEnvio(){
        return this.getValorDeclarado() * 1.5 + super.getCostoEnvio();
    }
    @Override
    public String getDescripcion(){
        return super.getDescripcion() + "entrega express";
    }
}
public class Fragil extends PaqueteDecorator{
    public Fragil (IPaquete wrappee){
        super(wrappee);
    }
    @Override
    public String getDescripcion(){
        return super.getDescripcion() + "fragil";
    }
}
```

```
public class Producto implements IPaquete{
    private String descripcion;
    private String destinatario;
    private String direccionDestino;
    private double valorDeclarado;
    public Producto(double valor, String desc, String dest, String dir){
        this.descripcion=desc;
        this.valorDeclarado=valor;
        this.direccionDestino=dir;
        this.destinatario=dest;
    }
    public String getDescripcion(){
        return this.descripcion;
    }
    public String getDestinatario(){
        return this.destinatario;
    }
    public String getDireccionDestino(){
        return this.direccionDestino;
    }
    public double getValorDeclarado(){
        return this.valorDeclarado;
    }
    public double getCostoEnvio(){
        return this.valorDeclarado * 0.05;
    }
}
```

- No es necesario dar una implementación completa de esta clase -

4. public class ProductoTest{
    private Producto prod;

    @BeforeEach
    void setUp(){
        prod = new Producto(20000, "Caja de libros", "x", "y");
    }
    @test
    void testAll{
        PaqueteDecorator seguro = new ConSeguro(prod);
        PaqueteDecorator express = new Express(seguro);
        assertEquals("Caja de libros con seguro entrega express", express.getDescripcion());

        assertEquals(15000, express.getCostoEnvio());
    }
}