UNIVERSIDAD
DE GRANADA

Análisis de Implementación de Patrones de Diseño

Luis Arrabal Gutiérrez
Jesús Delgado Ramos
Gerardo Bonet Pérez

Marzo 2025

Índice

1. Ejercicio 1: Patrón Abstract Factory en Java

1.1. Descripción del Ejercicio

El ejercicio consiste en simular dos carreras simultaneas, una de carretera y otra de montaña, con el mismo nunero inicial de bicilietas. Durante la ejecución, se deben retirar un 20 % de las que participan en la de montaña y un 10 % de las que participan en la de carretera. Además se deben ejecutar en paralelo teniendo una misma duración de 60 segundos.

1.2. Patrones Aplicados y Diagrama de Clases

En este ejercicio se han empleado los siguientes patrones de diseño:

- **Factoría Abstracta**: Se define la interfaz `FactoriaCarreraBicicleta`, que incluye métodos para crear carreras y bicicletas dependiendo si es de montaña o si es de carretera.
- **Factoría Método**: Se definen las clases que implementan la interfaz `FactoriaCarreraBicicleta` (`FactoriaMontaña` y `FactoriaCarretera`), las cuales implementan los métodos especificos de creación de carreras y bicicletas dependiendo del tipo que sean.

El diagrama de clases correspondiente es el siguiente:

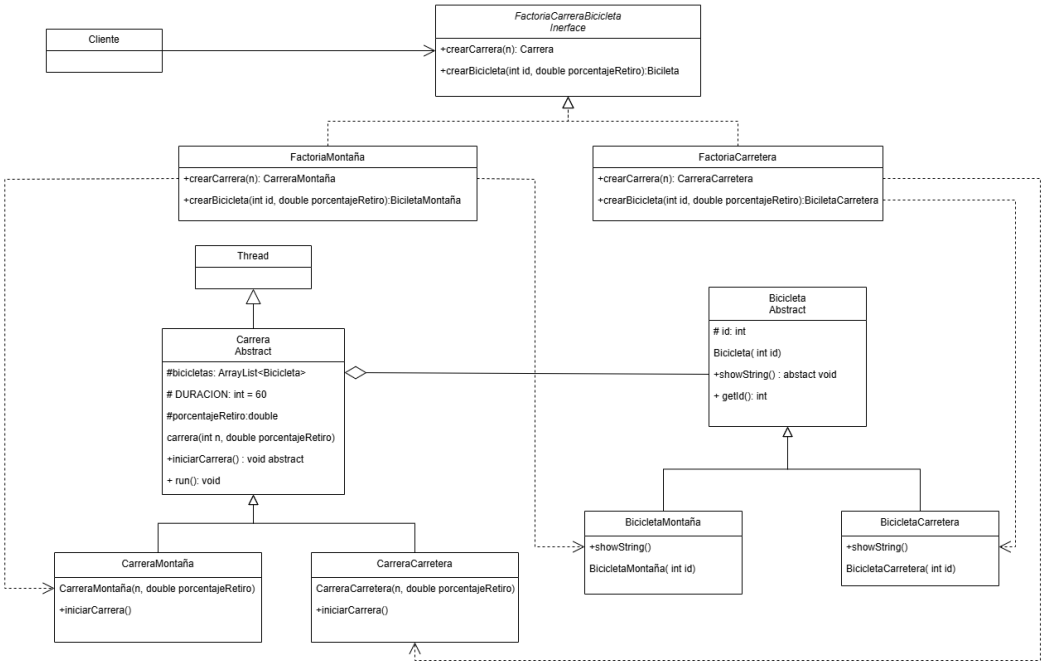

Figura 1: Diagrama UML del ejercicio 1

1.3. Implementación del Código de Hebras

Para la implementación de las hebras, he hecho que la clase `Carrera` extienda de la clase `Thread`. Esta sobrescribirá el método `run` de `Thread`, en el cual se llamará al método abstracto `iniciarCarrera`. Este método se encargará de calcular, para cada tipo de carrera, el número de bicicletas a retirar y de mostrar por pantalla, cada segundo durante la ejecución de la hebra, las bicicletas que continúan y las que se retiran.

Finalmente para el inicio de las hebras desde la clase `Cliente` se llama a el metodo `start()` de cada tipo de carrera.

Este seria el código de `iniciarCarrera` de la clase `CarreraCarretera`

```
@Override
    public void iniciarCarrera() {
        System.out.println("Iniciando carrera carretera");

        //Numero de bicicletas retiradas
        int numeroRetiradas = (int)(bicicletas.size() * porcentajeRetiro);
        System.out.println("Numero de bicicletas a retirar de la carrera de
            carretera: "+numeroRetiradas);

        //Contador de bicis retiradas
        int contadorRetiradas = 0;
        //Intervalo de tiempo de cada cuanto se retira una bici, si hay menos
            de 60 bicis a retirar cada 1 segundo
        int intervaloTiempo;
        if(numeroRetiradas > 60){
            intervaloTiempo = (int) (DURACION/numeroRetiradas)*1000;
        }
        else{
            intervaloTiempo = 1000;
        }


        //Guardo el instante de inicio
        long iniTime = System.currentTimeMillis();

        //Mido la diferencia de tiempo, lo divido por 1000 porque es en
            milisegundos
        while((System.currentTimeMillis()-iniTime)/1000<DURACION) {
            try{
                for(Bicicleta bici : bicicletas){
                    bici.showString();
                }

                Thread.sleep(intervaloTiempo);

                if(contadorRetiradas < numeroRetiradas && !bicicletas.isEmpty
                    ()){
                    Random rand = new Random();
                    int indice = rand.nextInt(bicicletas.size());
                    System.out.println("Bicileta con el id " + bicicletas.get(
                        indice).getId() + " se ha retirado de la carrera de
                        Carretera");
                    bicicletas.remove(indice);
                    contadorRetiradas++;
                }

            }catch (InterruptedException e) {
                System.out.println("La carrera de carretera fue interrumpida")
                    ;
```

```
                return;
            }

        }

        System.out.println("La carrera de carretera ha finalizado");
    }
```

1.4. Ejemplo de ejecución

Para esta ejecución he reducido el tiempo de las carreras a 10 segundos y he utilizado 10 bicicletas y este ha sido el resultado obtenido:

```
Ingrese el numero de bicicletas: 10
Iniciando la carrera de Monta a
Iniciando carrera carretera
Numero de bicicletas a retirar de la carrera de monta a: 2
Numero de bicicletas a retirar de la carrera de carretera: 1
 Bicicleta de monta a con ID: 0
 Bicicleta de carretera con ID: 0
 Bicicleta de carretera con ID: 1
 Bicicleta de carretera con ID: 2
 Bicicleta de carretera con ID: 3
 Bicicleta de carretera con ID: 4
 Bicicleta de carretera con ID: 5
 Bicicleta de carretera con ID: 6
 Bicicleta de carretera con ID: 7
 Bicicleta de carretera con ID: 8
 Bicicleta de carretera con ID: 9
 Bicicleta de monta a con ID: 1
 Bicicleta de monta a con ID: 2
 Bicicleta de monta a con ID: 3
 Bicicleta de monta a con ID: 4
 Bicicleta de monta a con ID: 5
 Bicicleta de monta a con ID: 6
 Bicicleta de monta a con ID: 7
 Bicicleta de monta a con ID: 8
 Bicicleta de monta a con ID: 9
Bicileta con el id 4 se ha retirado de la carrera de Monta a
Bicileta con el id 2 se ha retirado de la carrera de Carretera
```

En cada segundo muestra las bicicletas que estan en la carrera y si se retira alguna, cuando ya no quedan bicicletas por retirar se mantiene la carrera igual mostrando en cada segundo las bicicletas que han quedado hasta el final de la ejecución.

```
 Bicicleta de monta a con ID: 0
 Bicicleta de carretera con ID: 6
 Bicicleta de monta a con ID: 2
 Bicicleta de monta a con ID: 3
 Bicicleta de monta a con ID: 5
 Bicicleta de monta a con ID: 6
 Bicicleta de monta a con ID: 7
 Bicicleta de monta a con ID: 8
 Bicicleta de monta a con ID: 9
 Bicicleta de carretera con ID: 7
 Bicicleta de carretera con ID: 8
 Bicicleta de carretera con ID: 9
La carrera de monta a ha finalizado
La carrera de carretera ha finalizado
```

2. Ejercicio 2: Patrón Decorator en Python

2.1. Requisitos

- Integrar modelos de Hugging Face para resumir, traducir y ampliar texto.
- Implementar **Patrón Decorator** para añadir funcionalidades dinámicas.
- Configuración y paso de variables mediante JSON.

2.2. Patrones Aplicados

- **Decorator**: Los decoradores (`TranslationDecorator`, `ExpansionDecorator`) envuelven al componente `LLMDecorator` que a su vez envuelve al componente LLM.

2.3. Estructura Clave

```
Clase base abstracta

class LLM(ABC):
@abstractmethod
def generate_summary(...): ...

class BasicLLM(LLM):
    def generate_summary(...): ...

class LLMDecorator(LLM):
    def generate_summary(...): ...

Decorador de traduccion

class TranslationDecorator(LLMDecorator):
def init(self, llm, model):
    self._llm = llm # Composicion del componente base
def generate_summary(...):
    texto_traducido = self._translate(...)
    return self._llm.generate_summary(...)
def _translate(...):
    # Llamadas a la API de hugging face

Decorador de Ampliacion

class ExpansionDecorator(LLMDecorator):
def init(self, llm, model):
    self._llm = llm # Composicion del componente base
def generate_summary(...):
    texto_resumido = self._expand(...)
    return self._llm.generate_summary(...)
def _expand(...):
    # Llamadas a la API de hugging face
```

2.4. Ejemplo de ejecución

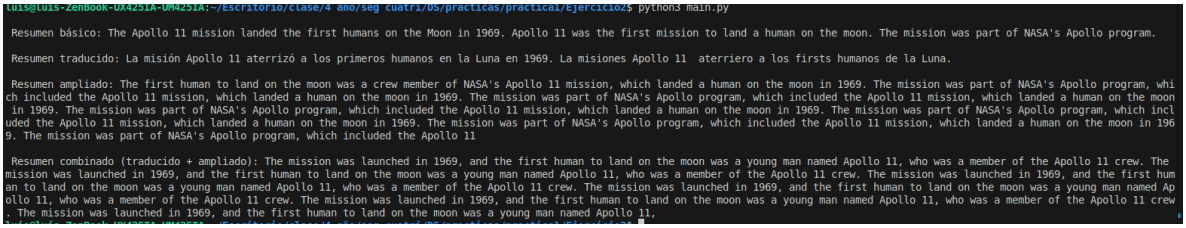

Figura 2: Salida terminal ejercicio 2

2.5. Observaciones y Diagrama de Clases

- Uso de herencia múltiple mediante `ABC( Abstract Base Class ).`
- Configuración mediante archivo JSON que permite flexibilidad.
- Se ha observado que el número de peticiones para realizar llamadas a la API es escaso.
- Se han detectado ocasiones en las que los modelos no funcionan de manera óptima, lo que podría afectar al rendimiento y la fiabilidad del sistema.

El diagrama de clases correspondiente es el siguiente:

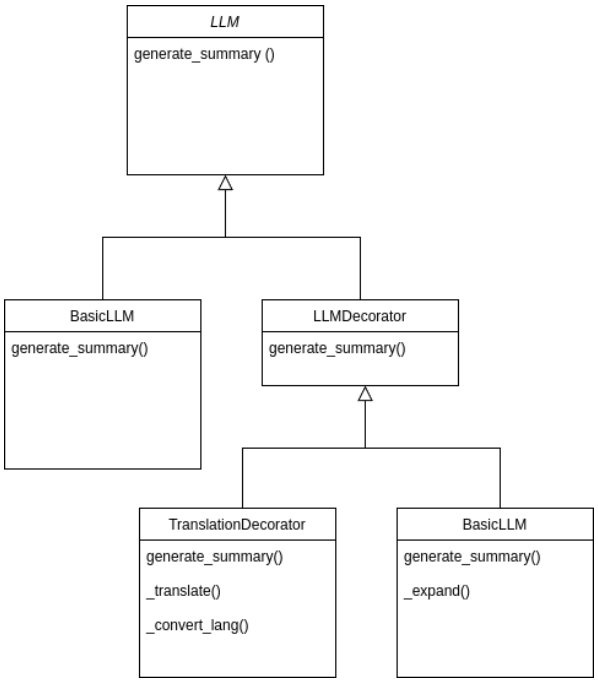

Figura 3: Diagrama UML del ejercicio 2

3. Ejercicio 3: Patrón Strategy en Python

3.1. Requisitos

- Extraer datos de 5 páginas de quotes.toscrape.com.
- Implementar dos estrategias: `BeautifulSoup` y `Selenium`.
- Guardar resultados en YAML.

3.2. Patrones Aplicados

- **Strategy**: Las clases `BeautifulSoupStrategy` y `SeleniumStrategy` implementan la interfaz `ScrapingStrategy`.

3.3. Estructura Clave

```
class ScrapingStrategy(ABC):
@abstractmethod
def scrape(self, url): ...

class Scraper:
def init(self, strategy: ScrapingStrategy):
    self.strategy = strategy # Inyeccion de dependencia

def get_quotes(self):
    for page in range(1,6):
        url = base_url.format(page)
        quotes = self.strategy.scrape(url)
        all_quotes.extend(quotes)

class BeautifulSoupStrategy(ScrapingStrategy):
    def scrape(self, url):
        for quote in soup.select(".quote"):
            text = quote.find("span", class_="text").get_text()
            author = quote.find("small", class_="author").get_text()
            tags = [tag.get_text() for tag in quote.find_all("a", class_="tag"
                )]
            quotes.append({"text": text, "author": author, "tags": tags})
            # Por cada cita almacenamos el texto, autor y tags

class SeleniumStrategy(ScrapingStrategy):
    def __init__(self):
        self.driver = webdriver.Chrome(service=Service("/usr/local/bin/
            chromedriver"), options=options) # Usamos el driver para
            automatizar la navegacion en el navegador

    def scrape(self, url):
        self.driver.get(url) # Abre la url en el navegador

        elements = self.driver.find_elements(By.CLASS_NAME, "quote") # Busca
            todas las citas

        for quote in elements:
            text = quote.find_element(By.CLASS_NAME, "text").text
            author = quote.find_element(By.CLASS_NAME, "author").text
            tags = [tag.text for tag in quote.find_elements(By.CLASS_NAME, "
                tag")]
            quotes.append({"text": text, "author": author, "tags": tags})
            # Igual que antes almacenamos todas las citas
```

```
    def __end__(self):
        self.driver.quit() # Cerramos el driver
```

3.4. Ejemplo de ejecución

```
(.venv) jesus@jesus-HP-Pavilion-Gaming-Laptop-16-a0xxx:~/Escritorio/DS/PRÁCTICAS/P1/EJERCICIO 3$ python3 scraping.py
Seleccione la estrategia de scraping:
1 - BeautifulSoup
2 - Selenium
Ingrese el número de la opción deseada: 1
Scraping completado. Se guardaron 50 citas en 'quotes.yaml'.
```

Figura 4: Salida terminal ejercicio 3

```
- author: Albert Einstein
  tags:
  - change
  - deep-thoughts
  - thinking
  - world
  text: “The world as we have created it is a process of our thinking. It cannot be
    changed without changing our thinking.”
- author: J.K. Rowling
  tags:
  - abilities
  - choices
  text: “It is our choices, Harry, that show what we truly are, far more than our
    abilities.”
- author: Albert Einstein
  tags:
  - inspirational
  - life
  - live
  - miracle
  - miracles
  text: “There are only two ways to live your life. One is as though nothing is a
    miracle. The other is as though everything is a miracle.”
```

Figura 5: Forma de la que se guardan las 50 citas en quotes.yaml

3.5. Observaciones y diagrama de clase

- Uso de `Options().add_argument(-headless")` en Selenium optimiza recursos.
- El formato YAML garantiza legibilidad y estructura jerárquica.

El diagrama de clases correspondiente es el siguiente:

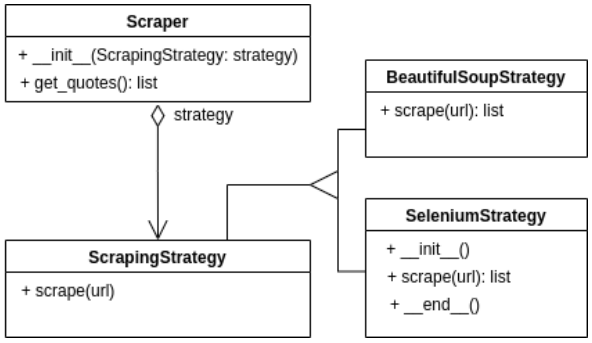

Figura 6: Diagrama UML del ejercicio 3

4. Ejercicio 4: Patrón Filtros de Intercepción en Java

4.1. Requisitos

- Validar correo (dominio gmail/hotmail) y contraseña (3 filtros personalizados).
- Implementar cadena de filtros sin modificar clases existentes.

4.2. Patrones Aplicados

- **Chain of Responsibility**: Los filtros (`FiltroCorreo`, `FiltroLongitud`, etc.) se encadenan en `CadenaFiltros`.

4.3. Estructura Clave

```
interface Filtro {
boolean ejecutar(String dato);
}

class CadenaFiltros {
private List<Filtro> filtros = new ArrayList<>();
public boolean ejecutar(String dato) {
for (Filtro filtro : filtros) {
if (!filtro.ejecutar(dato)) return false;
}
return true;
}
}
```

4.4. Ejemplo de ejecución

```
Ingrese su correo: @gmail.com
Correo inválido: falta texto antes de '@'
Ingrese su correo: ds@
Correo inválido: falta texto después de '@'
Ingrese su correo: ds@hola.com
Correo inválido: dominio no permitido (solo gmail.com o hotmail.com)
Ingrese su correo: ds@gmail.com
Ingrese su contraseña: abcd
Contraseña inválida: debe tener al menos 8 caracteres
Ingrese su contraseña: abcdefgh
Contraseña inválida: debe contener al menos un número
Ingrese su contraseña: 1abcdefg
Contraseña inválida: debe contener al menos una letra mayúscula
Ingrese su contraseña: 1Abcdefg
Autenticación exitosa
```

Figura 7: Ejecución del ejercicio 4

4.5. Observaciones y diagrama de clases

- Separación clara entre validación de correo y contraseña.
- Patrón de diseño : se utiliza la cadena de responsabilidad, lo que permite agregar nuevos filtros fácilmente sin modificar el flujo principal.
- Uso de expresiones regulares para validación de contraseña.

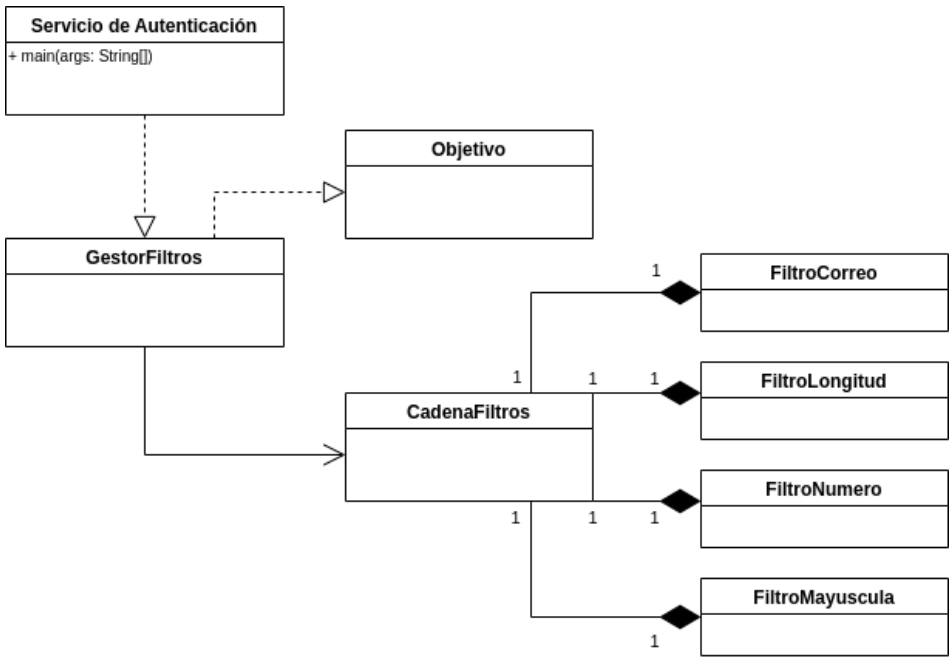

Figura 8: Diagrama UML del ejercicio 4

5. Enlaces adicionales

Enlace al repositorio de la asignatura: **Github**