

Desarrollo de Software Práctica 1



UNIVERSIDAD DE GRANADA

Esther Prats Hodar
Germán Álvarez Gavilán
Lorena Ramos Ruiz

20 de marzo de 2025

Índice

1. Introducción	2
2. Ejercicio 1	3
2.1. Explicación	3
2.2. Ejemplos de ejecución	4
2.3. Diseño diagrama UML	4
3. Ejercicio 2	5
3.1. Explicación	5
3.2. Ejemplo de ejecución	6
3.3. Diseño diagrama UML	7
4. Ejercicio 3	8
4.1. Explicación	8
4.2. Ejemplo de ejecución	9
4.3. Diseño diagrama UML	10
5. Ejercicio 4	11
5.1. Explicación	11
5.2. Ejemplo de ejecución	11
5.3. Diseño diagrama UML	12

1. Introducción

En esta Práctica se desarrollan e implementan soluciones software basadas en los patrones de diseño vistos en clase usando tanto Java como Python.

Estos ejercicios nos han sido muy útiles para practicar la usabilidad de estos patrones y nos han ayudado a entender en una mayor profundidad su funcionamiento.

Nos han parecido muy interesantes los ejercicios 2 y 3 ya que nunca antes habíamos hecho este tipo de proyectos en la carrera y creemos que ha sido un acierto poner estos ejercicios, ya que gracias a ellos hemos descubierto cómo realizar ciertas cosas que no sabíamos hacer.

A continuación vamos a explicar cada implementación de los ejercicios.

2. Ejercicio 1

2.1. Explicación

El objetivo de este ejercicio es implementar en Java un programa que simule dos carreras (de montaña y de carretera) usando los patrones de diseño Factoría Abstracta y Método Factoría. Las dos carreras se ejecutan simultáneamente con hebras. Sobre nuestra implementación:

Bicicleta y carrera son las clases abstractas de las cuales salen cada tipo de Bicicleta y de Carrera: BicicletaMontaña, BicicletaCarretera, CarreraMontaña y CarreraCarretera. Hay que destacar la función de las Carreras que lo que hace es retirar un porcentaje de bicicletas al terminar la carrera (está establecido en el enunciado). Cada una de estas tiene su constructor.

Por otro lado tenemos las factorías: FactoriaCarrera y FactoriaBicicleta la cual implementa FactoriaMontaña y FactoriaCarretera. Cada una crea las Bicicletas y Carreras.

La simultaneidad de las Carreras está implementada con hilos (Thread). Cada hilo crea una instancia de su factoría correspondiente para generar las bicicletas y carrera. Además usamos Thread.sleep para que la simulación dure 60 segundos. Vemos el ejemplo de la Carrera de Montaña, y la de Carretera es similar. Para gestionar el número de Bicicletas que se retiran en cada caso, dentro de la clase Carrera hay un atributo retirados que en cada caso, al crear la carrera se le establece un valor u otro.

```
Thread hiloMontana = new Thread() {
    @Override
    public void run() {
        FactoriaCarreraBicicleta factoria = new FactoriaMontana();
        Carrera carrera = factoria.crearCarrera();

        for(int i = 0; i < N; i++){
            carrera.bicicletas.add(factoria.crearBicicleta());
        }

        System.out.println("Carrera de Montaña iniciada con " + carrera.bicicletas.size());
        try{
            Thread.sleep(60000);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }

        carrera.retirarBicicletas();
        System.out.println("Carrera de Montaña terminada con " + carrera.bicicletas.size());
    }
};
```

```
hiloMontana.start();
hiloCarretera.start();
```

El número de bicicletas inicial es elegido aleatoriamente al inicio de la ejecución (un número del 5 al 20).

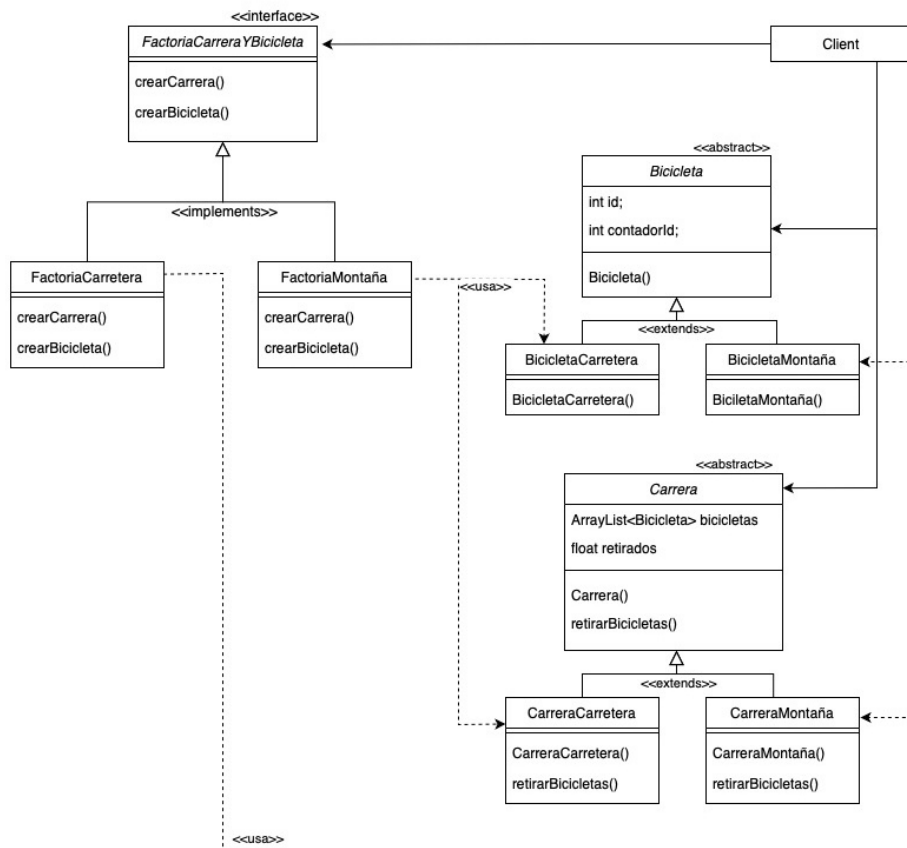
Más abajo tenemos un ejemplo de salida de la ejecución.

2.2. Ejemplos de ejecución

```
Número de bicicletas inicial: 11
Carrera de Montaña iniciada con 11
Carrera de Carretera iniciada con 11
Carrera de Carretera terminada con 9
Carrera de Montaña terminada con 8
-----
BUILD SUCCESS
-----
Total time: 01:00 min
Finished at: 2025-03-13T21:52:44+01:00
-----
```

```
Número de bicicletas inicial: 16
Carrera de Carretera iniciada con 16
Carrera de Montaña iniciada con 16
Carrera de Montaña terminada con 12
Carrera de Carretera terminada con 14
-----
BUILD SUCCESS
-----
Total time: 01:00 min
Finished at: 2025-03-13T21:58:34+01:00
-----
```

2.3. Diseño diagrama UML



3. Ejercicio 2

3.1. Explicación

Este ejercicio consiste en desarrollar un programa en Python que utilice la API de Hugging Face para generar resúmenes de texto, traducirlos y ampliarlos mediante el patrón de diseño Decorator. La finalidad del ejercicio es que aprendamos a implementar el patrón Decorator para añadir funcionalidades de manera modular y configurable, mientras interactuamos con modelos de IA sin necesidad de descargarlos localmente.

Para ello, poseemos un archivo cuya extensión es .json, dentro del cual encontramos los parámetros de entrada necesarios para realizar cada una de las funcionalidades, entre ellas, destaca el texto, el idioma de entrada, el idioma de salida y el modelo a escoger. Además del modelo de traducción y expansión.

El programa lee este archivo y ya se decide que decoradores aplicar dinámicamente. Se muestra el archivo:

```
{ } entrada.json > ...
1  {
2    "texto": "Hello, how are you? I would like to know if you can help me with a problem I have with my computer.",
3    "input_lang": "en",
4    "output_lang": "es",
5    "model_llm": "facebook/bart-large-cnn",
6    "model_translation": "Helsinki-NLP/opus-mt-en-es",
7    "model_expansion": "facebook/blenderbot-400M-distill"
8  }
9
```

Un punto a tener en cuenta es que, el acceso a la API, se hace a través de un token, el cual ha debido de ser creado anteriormente y estar asociado a una cuenta. Para hacer esto más personal, ya que el token es privado, he introducido el token mediante otro archivo con extensión .json. Así, se tiene libertad para sustituirlo, por el propio de cada uno cuando se desee. Además, declaro una variable **HEADERS**, que tiene la autorización del token, véase también en la imagen adjunta:

```
0  with open("token.json", "r") as file:
1      config = json.load(file)
2
3  API_KEY = config["HUGGINGFACE_API_KEY"]
4  HEADERS = {"Authorization": f"Bearer {API_KEY}", "Content-Type": "application/json"}
```

Quedando el archivo .json que contiene al token de la siguiente forma:

```
{ } token.json > abc HUGGINGFACE_API_KEY
1  {
2    "HUGGINGFACE_API_KEY": "
3  }
```

Para la implementación del ejercicio, primero se define LLM, una clase abstracta que define la interfaz para los modelos de lenguaje. Obliga a que cualquier clase que herede de ella implemente **generate summary**.

Luego tenemos BasicLLM, que implementa **generate summary**, que envía el texto a la API de Hugging Face para que el modelo **model llm** genere un resumen. Además, se incluye el manejo de errores en la API.

Después, se implementa la clase base decoradora LLMDecorator, que extienden la funcionalidad de BasicLLM sin modificar el código. Es un decorador que envuelve un objeto LLM, y no se altera su comportamiento, solo lo reenvuelve.

Tenemos dos decoradores:

- TranslationDecorator
- ExpansionDecorator

El TranslationDecorator, genera un resumen con el LLM base, y luego lo traduce usando el modelo de traducción de Hugging Face.

El ExpansionDecorator, genera un resumen con el LLM base, y luego lo expande usando el modelo de expansión de Hugging Face.

El main se encarga de: cargar los parámetros desde el archivo **entrada.json**, crear una instancia de BasicLLM para generar un resumen básico, usar TranslationDecorator para traducir el resumen, usar ExpansionDecorator para expandirlo, aplicar ambos decoradores para primero traducir y luego expandir, y por último mostrar todos los resultados en consola.

3.2. Ejemplo de ejecución

```
(venv) lorenarr2004@MacBook-Air-de-Lorena ejer2 % python3 ejer2.py
Iniciando la función main...
Archivo JSON cargado correctamente.
Texto a procesar: Artificial intelligence (AI) is transforming industries worldwide. Businesses use AI for automation, data analysis, and customer service. Machine learning models, such as neural networks, enable computers to recognize patterns and make predictions. However, ethical concerns arise regarding data privacy and job displacement. As AI technology advances, it is crucial to develop policies that ensure its responsible use while maximizing its benefits for society.

Resumen básico:
Artificial intelligence (AI) is transforming industries worldwide. Businesses use AI for automation, data analysis, and customer service. However, ethical concerns arise regarding data privacy and job displacement. As AI technology advances, it is crucial to develop policies that ensure its responsible use.

Resumen traducido:
La inteligencia artificial (AI) está transformando las industrias en todo el mundo. Las empresas utilizan la IA para la automatización, el análisis de datos y el servicio al cliente. Sin embargo, surgen preocupaciones éticas con respecto a la privacidad de los datos y el desplazamiento de puestos de trabajo.

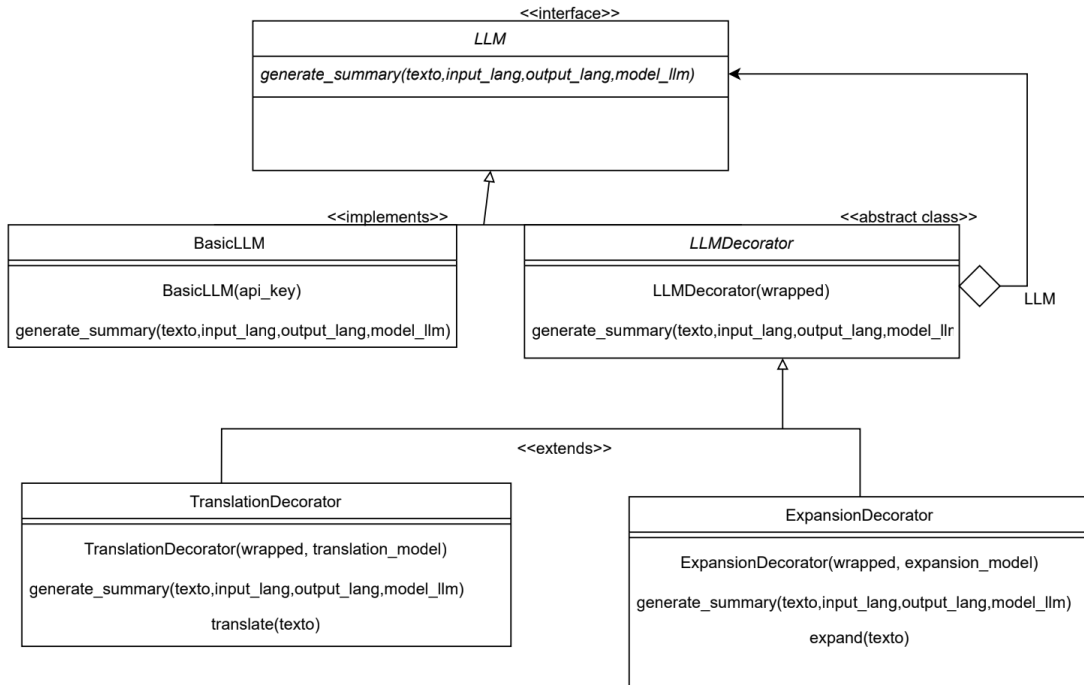
Resumen expandido:
Yes, I agree with you. I think it is important for companies to have an understanding of the importance of technology.

Resumen traducido y expandido:
Las Vegas is the most populous city in the state of Nevada.
```

Como vemos la ejecución es simple, basta con hacer **python3 ejer2.py** (en mi caso) o **python ejer2.py** , hay que tener en cuenta las bibliotecas usadas, en mi caso, dentro de un entorno virtual, para evitar conflictos con otros paquetes.

Las bibliotecas usadas son json, **requests**(para hacer solicitudes HTTP a la API de Hugging Face) y **transformers**(para interactuar con modelos de Hugging Face).

3.3. Diseño diagrama UML



4. Ejercicio 3

4.1. Explicación

Este ejercicio pretende sacar las citas y sus datos de la página `quotes.toscrap.com`, para ello se usarán las bibliotecas de Python de Selenium y BeautifulSoup. Para su implementación se ha usado el patrón Strategy, cuya implementación se encuentra en el archivo `patronDiseño.py` y para la que se ha creado una clase abstracta `Scraper`, que define el método abstracto `scrapear`, que recibe una web. De esta clase heredan tanto `SeleniumScraper` como `BeautifulSoupScraper`. Además, para controlar la estrategia a utilizar se ha creado la clase `Estrategia`, que tiene un constructor en el que se le pasa un `Scraper` que se guarda en el atributo `estrategia` del objeto, además tiene un método `scrapear`, que recibe una web y llama al método `scrapear` del `Scraper` guardado en `estrategia`.

En el caso de Selenium, para la implementación de `scrapear`, se usa su `webdriver` para abrir Firefox y obtener la web, luego repetimos `n` veces los siguientes pasos, siendo `n` el número de páginas a `scrapear`. Realizamos una pausa para que cargue la página, ya que si no ha cargado antes de empezar a buscar datos, no los encuentra. A partir de ahí, buscamos los datos como se explica más tarde para poder devolverlos. Por último, en las `n-1` primeras páginas, buscamos por XPATH el botón de siguiente, dentro de este XPATH se ha añadido `li[@class='next']/a`, ya que si no se especifica la clase, devuelve el primero que encuentre, y a partir de la segunda página este es el botón de anterior, por lo que hace un bucle entre las páginas 1 y 2. Una vez ha encontrado el botón, hace click en él, lo que supone cambiar de página, en caso de no encontrar el botón, se devuelve una excepción con el mensaje “No se encontró el botón o hubo un error”

Por otro lado, para BeautifulSoup, dentro de la implementación del método `scrapear`, usamos `request.get(url)`, que nos devuelve el código html de la página, a partir del cuál buscamos y extraemos los datos para poder devolverlos. En este caso no es posible pulsar botones, ya que no estamos en el navegador, si no que busca directamente en el código html. Para poder extraer citas de distintas páginas simulamos las pulsaciones en el botón de Siguiente cambiando la url dentro del bucle, siendo esta la url base, seguida de `/page/` y el número de página a `scrapear`.

La búsqueda de los elementos necesarios es muy similar en ambos casos, de hecho, el único cambio es la función usada. Para encontrar los datos empezamos localizándolos el elemento cuya clase es “quote”, para cada elemento, buscamos los atributos cuya clase es “text” (la cita), “author” (el autor) y “tag” (las etiquetas). Una vez tenemos los datos, en la variable `tags` unimos el texto de todas las etiquetas que tenemos para convertirlo en uno solo. Una vez que tenemos estos 4 textos, los añadimos en la variable `datos`, usando una estructura tipo diccionario.

El programa principal se encuentra en el archivo `main.py` y consta de un menú con 3 opciones numéricas, en el caso de la opción 1, se usa el scraper de Selenium, en el caso de la 2, el de BeautifulSoup y con la tercera opción se cierra el programa. En el caso de las dos primeras, se genera una estrategia y se elige la carpeta en la que se va a guardar el archivo YAML que se generará, pudiendo esta ser `Resultados/Selenium` o `Resultados/BeautifulSoup`. Seguido de esto, si no se ha decidido cerrar el programa, se hace el scrapeo de la página usando la estrategia elegida, si se sacan datos, se crea la carpeta necesaria en caso de que no exista y se genera el nombre del archivo y la ruta del mismo. Se abre el archivo y se introducen todos los datos. Por último se imprime un mensaje de confirmación con la ruta en la que se ha guardado y se vuelve al menú principal.


4.2. Ejemplo de ejecución

```
(DDSI) gag_04@eduroam060023 ej3 % python3 main.py

Introduce 1 para usar Selenium, 2 para usar BeautifulSoup y 3 para salir del programa: 1
Datos guardados en /Users/gag_04/Documents/Académico/Universidad/3º/Segundo Cuatrimestre/DS/ej3/Resultados/Selenium/citas_1.yaml.

Introduce 1 para usar Selenium, 2 para usar BeautifulSoup y 3 para salir del programa: 2
Datos guardados en /Users/gag_04/Documents/Académico/Universidad/3º/Segundo Cuatrimestre/DS/ej3/Resultados/BeautifulSoup/citas_2.yaml.

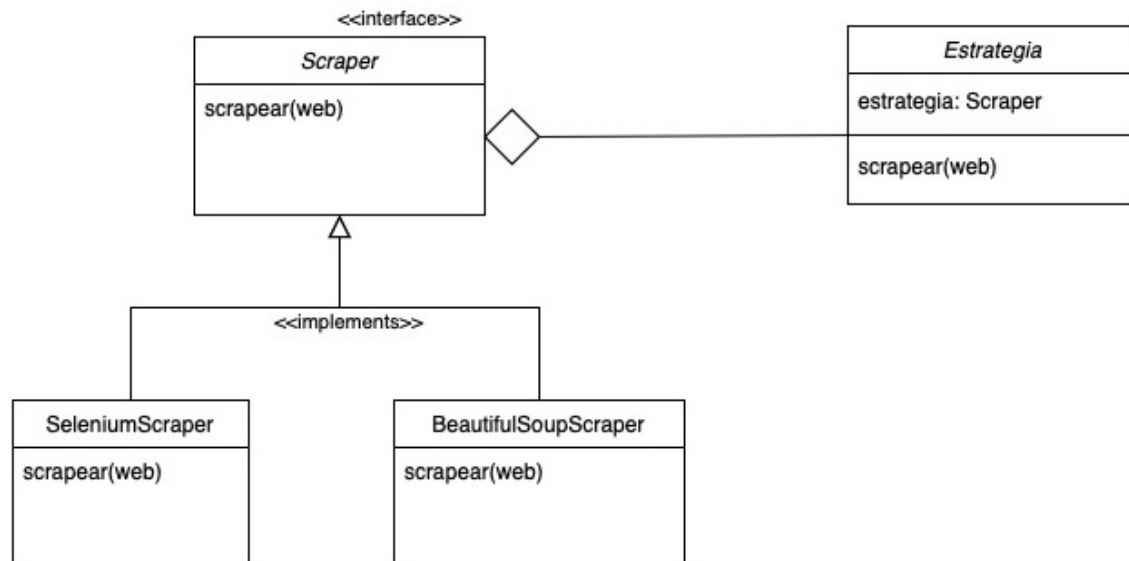
Introduce 1 para usar Selenium, 2 para usar BeautifulSoup y 3 para salir del programa: 3
El programa ha sido finalizado correctamente.
(DDSI) gag_04@eduroam060023 ej3 %
```



The screenshot shows a VS Code editor window with the file 'citas_2.yaml' open. The file contains three entries, each with an ID, a quote, an author, and a list of tags.

```
- ID: 1
  Cita: "The world as we have created it is a process of our thinking. It cannot be
    changed without changing our thinking."
  Autor: Albert Einstein
  Etiquetas:
    - change
    - deep-thoughts
    - thinking
    - world
- ID: 2
  Cita: "It is our choices, Harry, that show what we truly are, far more than our
    abilities."
  Autor: J.K. Rowling
  Etiquetas:
    - abilities
    - choices
- ID: 3
  Cita: "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."
  Autor: Albert Einstein
  Etiquetas:
    - inspirational
    - life
    - live
    - miracle
    - miracles
```

4.3. Diseño diagrama UML



5. Ejercicio 4

5.1. Explicación

El objetivo de este ejercicio es implementar, en nuestro caso en Java, un programa para validar las credenciales de un usuario usando para ello el patrón de diseño Filtros de Intercepción. Este patrón lo usamos para validar los datos ingresados. Sobre nuestra implementación:

Filter (filtros), son los responsables de realizar las validaciones de los datos de entrada. Los filtros que tenemos son: EmailFilter que valida que el correo sea de la forma nombre@dominio siendo dominio gmail.com o hotmail.com, LengthFilter encargado de comprobar que la contraseña tiene al menos 8 caracteres, NumbersFilter encargado de comprobar que la contraseña tiene al menos 2 números y SpecialCharacterFilter que comprueba que la contraseña tenga al menos 1 carácter especial.

FilterChain administra el conjunto de filtros y si alguno falla el procesamiento se detiene.

FilterManager controla la cadena de filtros y permite añadir los filtros que necesites. Asegura que los datos de entrada pasen correctamente las validaciones.

Target expresa que la entrada satisface las validaciones una vez aplicados los filtros.

Client interactúa con el FilterManager para hacer la request de validar email y contraseña.

5.2. Ejemplo de ejecución

```
Introduce tu correo:
esther@correo.ugr.es
Introduce tu contraseña:
esther20

Validando el correo...
ERROR: El dominio debe ser gmail.com o hotmail.com.
Algún filtro ha fallado, intercepción terminada.
El correo no ha pasado las validaciones correspondientes.
-----
BUILD SUCCESS
-----

Introduce tu correo:
esther
Introduce tu contraseña:
55hola!!

Validando el correo...
ERROR: El correo no contiene @.
Algún filtro ha fallado, intercepción terminada.
El correo no ha pasado las validaciones correspondientes.
-----
BUILD SUCCESS
-----
```

```
Introduce tu correo:
esther@gmail.com
Introduce tu contraseña:
hola2hola7

Validando el correo...
Validación exitosa.

Validando el contraseña...
ERROR: La contraseña debe obtener al menos un carácter especial.
Algún filtro ha fallado, intercepción terminada.
La contraseña no ha pasado las validaciones correspondientes.
-----
BUILD SUCCESS
-----

Introduce tu correo:
esther@gmail.com
Introduce tu contraseña:
holahola!2

Validando el correo...
Validación exitosa.

Validando el contraseña...
ERROR: La contraseña debe contener al menos 2 números.
Algún filtro ha fallado, intercepción terminada.
La contraseña no ha pasado las validaciones correspondientes.
-----
BUILD SUCCESS
-----
```

5.3. Diseño diagrama UML

