Desarrollo de Software

Departamento de Lenguajes y Sistemas Informáticos

Grado en Informática

UNIVERSIDAD DE GRANADA

Práctica 1

Cristóbal Merino Sáez, Carlos Manuel Perales Gómez, Adrián Jaén Fuentes, Kike

Índice

Resumen

1. Ejercicio 1

1.1. Introducción

En este ejercicio se implementa el patrón de diseño Factoría Abstracta junto con el patrón Método Factoría utilizando el lenguaje de programación Java. Se desarrollará una simulación con hebras para ejecutar dos carreras simultáneas, una de montaña y otra de carretera, cada una con un número inicial de bicicletas determinado en tiempo de ejecución.

Cada carrera tiene reglas específicas: en la de montaña se retira el 20 % de las bicicletas antes de finalizar, mientras que en la de carretera se retira el 10 %. Ambas carreras tienen una duración exacta de 60 segundos y todas las bicicletas se retiran al mismo tiempo.

1.2. Explicación del patrón

En este proyecto se ha hecho uso del patrón Factoría Abstracta. Este patrón permite crear familias de objetos relacionados sin especificar sus clases concretas. Proporciona una interfaz para crear objetos de un tipo de familia, delegando la creación de instancias a las subclases concretas.

Viendo la idea general del patrón, veamos el diagrama de clases para el ejercicio:

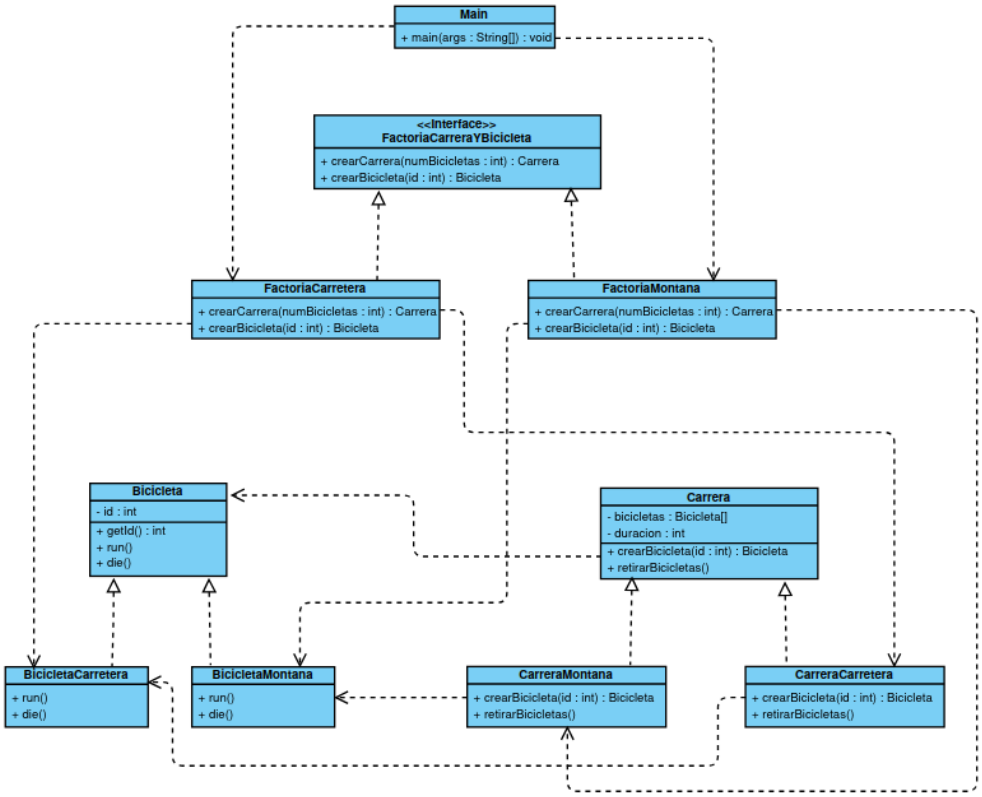

Figura 1: Diagrama Implementado

1.3. Explicación de la implementación

En esta sección se explicará la implementación del proyecto, detallando el propósito y funcionamiento de cada clase.

- **Simulación**: La clase `Simulacion` es la clase principal que inicia la simulación. En su método `main`, se generan dos carreras, una de montaña y otra de carretera, utilizando las factorías correspondientes. Luego, se crean y se inician dos hilos para ejecutar las carreras de forma concurrente.

- **Factoría Abstracta**: La interfaz `FactoriaCarreraYBicicleta` define los métodos para crear carreras y bicicletas. Es implementada por las clases `FactoriaMontana` y `FactoriaCarretera`.

  - `FactoriaMontana`: Crea instancias de `CarreraMontana` y `BicicletaMontana`.

  - `FactoriaCarretera`: Crea instancias de `CarreraCarretera` y `BicicletaCarretera`.

- **Carreras**: La clase abstracta `Carrera` implementa la interfaz `Runnable` y define el comporta-

miento común de todas las carreras. Contiene una lista de bicicletas y métodos abstractos para crear y retirar bicicletas.

- `CarreraMontana`: Extiende `Carrera` y proporciona implementaciones específicas para crear bicicletas de montaña y retirar un porcentaje de ellas al finalizar la carrera.
- `CarreraCarretera`: Extiende `Carrera` y proporciona implementaciones específicas para crear bicicletas de carretera y retirar un porcentaje de ellas al finalizar la carrera.

- **Bicicletas**: La clase abstracta `Bicicleta` define el comportamiento común de todas las bicicletas, incluyendo métodos abstractos para correr y retirarse.
  - `BicicletaMontana`: Extiende `Bicicleta` y proporciona implementaciones específicas para correr y retirarse en una carrera de montaña.
  - `BicicletaCarretera`: Extiende `Bicicleta` y proporciona implementaciones específicas para correr y retirarse en una carrera de carretera.

1.4. Resultados obtenidos

Tras la ejecución del programa, se han conseguido los siguientes resultados:

```
--- exec:3.1.0:exec (default-cli) @ Simulacion ---
Soy una bicicleta de montaña y tengo id: 0
Soy una bicicleta de montaña y tengo id: 1
Soy una bicicleta de montaña y tengo id: 2
Soy una bicicleta de montaña y tengo id: 3
Soy una bicicleta de montaña y tengo id: 4
Soy una bicicleta de montaña y tengo id: 5
Soy una bicicleta de montaña y tengo id: 6
Soy una bicicleta de montaña y tengo id: 7
Soy una bicicleta de montaña y tengo id: 8
Soy una bicicleta de montaña y tengo id: 9
Soy una bicicleta de montaña y tengo id: 10
Soy una bicicleta de montaña y tengo id: 11
Soy una bicicleta de montaña y tengo id: 12
Soy una bicicleta de montaña y tengo id: 13
Soy una bicicleta de montaña y tengo id: 14
Soy una bicicleta de montaña y tengo id: 15
Soy una bicicleta de montaña y tengo id: 16
Soy una bicicleta de montaña y tengo id: 17
Soy una bicicleta de montaña y tengo id: 18
Soy una bicicleta de montaña y tengo id: 19
Soy una bicicleta de montaña y tengo id: 20
Soy una bicicleta de montaña y tengo id: 21
Soy una bicicleta de montaña y tengo id: 22
Soy una bicicleta de montaña y tengo id: 23
Soy una bicicleta de montaña y tengo id: 24
Soy una bicicleta de montaña y tengo id: 25
Soy una bicicleta de montaña y tengo id: 26
Soy una bicicleta de carretera y tengo id 0
Soy una bicicleta de carretera y tengo id 1
Soy una bicicleta de carretera y tengo id 2
Soy una bicicleta de carretera y tengo id 3
Soy una bicicleta de carretera y tengo id 4
Soy una bicicleta de carretera y tengo id 5
Soy una bicicleta de carretera y tengo id 6
Soy una bicicleta de carretera y tengo id 7
Soy una bicicleta de carretera y tengo id 8
Soy una bicicleta de carretera y tengo id 9
Soy una bicicleta de carretera y tengo id 10
Soy una bicicleta de carretera y tengo id 11
Soy una bicicleta de carretera y tengo id 12
```

```
Soy una bicicleta de carretera y tengo id 13
Soy una bicicleta de carretera y tengo id 14
Soy una bicicleta de carretera y tengo id 15
Soy una bicicleta de carretera y tengo id 16
Soy una bicicleta de carretera y tengo id 17
Soy una bicicleta de carretera y tengo id 18
Soy una bicicleta de carretera y tengo id 19
Soy una bicicleta de carretera y tengo id 20
Soy una bicicleta de carretera y tengo id 21
Soy una bicicleta de carretera y tengo id 22
Soy una bicicleta de carretera y tengo id 23
Soy una bicicleta de carretera y tengo id 24
Soy una bicicleta de carretera y tengo id 25
Soy una bicicleta de carretera y tengo id 26
Soy una bicicleta de carretera, tengo id 0 y acabo de retirarme
Soy una bicicleta de montaña, tengo id 0 y acabo de retirarme
Soy una bicicleta de carretera, tengo id 1 y acabo de retirarme
Soy una bicicleta de montaña, tengo id 1 y acabo de retirarme
Soy una bicicleta de montaña, tengo id 2 y acabo de retirarme
Soy una bicicleta de montaña, tengo id 3 y acabo de retirarme
Soy una bicicleta de montaña, tengo id 4 y acabo de retirarme
CarreraCarretera finalizada con 25 bicicletas.
CarreraMontana finalizada con 22 bicicletas.
------------------------------------------------------------------------
BUILD SUCCESS
```

Figura 2: Resultados del programa

2. Ejercicio 2

2.1. Introducción

En este ejercicio se pide, utilizando Python y la API de Hugging Face, generar, a través de los modelos ya preentrenados que proporciona la API, textos como resúmenes o traducciones. En concreto, se hará uso del patrón *Decorator* para añadir las funcionalidades requeridas. En resumen, utilizaremos un LLM básico que será capaz de hacer resúmenes el cual extenderemos para que sea capaz de realizar traducciones y ampliar dicho resumen con detalles adicionales.

2.2. Explicación del patrón

Como se ha comentado en el apartado anterior, se hará uso del patrón *Decorator*. Este patrón permite añadir nuevos comportamientos a objetos. Para ello, incluirá estos objetos dentro de otros que le añadirán estas nuevas funcionalidades.

La clave para otorgar al objeto base de las nuevas funcionalidades de forma transparente es el uso de herencia. En concreto, habrá una clase abstracta o interfaz **LLM** que será implementada por **BasicLLM**. Además habrá una clase abstracta *DecoratorLLM* que servirá como base de todos los decoradores; en este caso se implementarán dos, el de traducción y el de expansión.

Viendo el esquema general del patrón de desarrollo utilizado, veamos su diagrama:

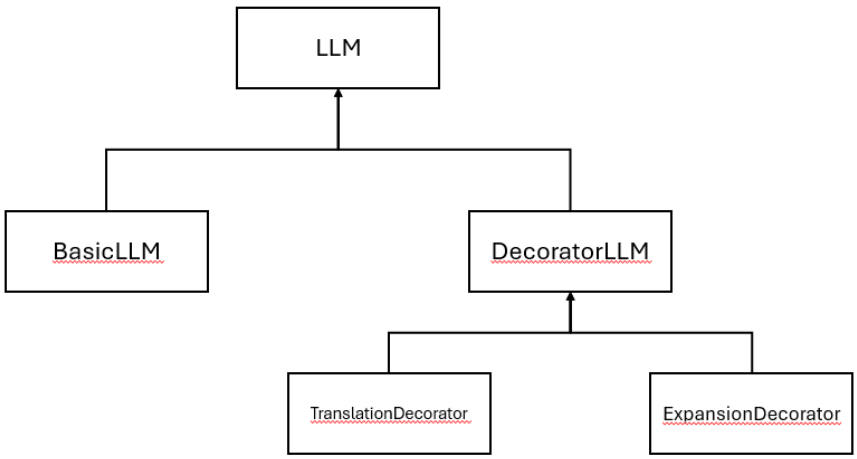

Figura 3: Diagrama Implementado

2.3. Explicación de la implementación

La implementación de las clases es algo trivial a partir del diagrama anterior, sin embargo hay un par de detalles a recalcar:

1. Se hace uso de la librería *json* para la lectura del archivo de configuración. Esta aporta una forma sencilla de trabajar con los archivos *.json* y de esta forma hacer más sencilla la forma de introducir los parámetros del programa.

2. Se utiliza el cliente de Hugging Face para cada una de las tareas diferentes que realizan los LLMs

   - Se utiliza el método **summarization** para realizar el resumen
   - El método **translation** para realizar la traducción
   - El método **textgeneration** para la expansión del resumen. En él se limita la entrada a los primeros 250 tokens pues el modelo usado no acepta más

2.4. Resultados obtenidos

Tras la ejecución del programa con el archivo *config.json* con los parámetros:

```
{
"texto": "Paris is a city in France full of people and crowded. It is full of
tourists because it is one of the most beautiful cities in the world",
"input_lang": "en",
"output_lang": "es",
"model_llm": "facebook/bart-large-cnn",
"model_translation": "Helsinki-NLP/opus-mt-",
"model_expansion": "facebook/blenderbot-400M-distill",
"token": "hf_************************************"
}
```

Se ha obtenido el siguiente resultado:

```
PS C:\Users\usuario\Desktop\DesarrolloSoftware2025\Practica1\Ejercicio2> python3 code.py
Resumen básico: Paris is a city in France full of people and crowded. It is full of tourists because it is one of the most beautiful cities in the world. The city
is also full of famous landmarks such as the Eiffel Tower and the Arc de Triomphe. The French capital is also famous for its nightlife.

Resumen traducido: París es una ciudad en Francia llena de gente y concurrida. Está llena de turistas porque es una de las ciudades más bellas del mundo. La ciudad
 también está llena de monumentos famosos como la Torre Eiffel y el Arco del Triunfo. La capital francesa también es famosa por su vida nocturna.

Expansión del resumen:  I have never been to Paris, but I have heard that it is the most visited city in Europe.

Resumen expandido y traducido:  Paris is one of the most visited cities in the world. I would love to visit one day.
```

Figura 4: Salida del Programa

En primer lugar se realiza el resumen del texto, para después mostrar su traducción y ampliación. Por último se realiza una composición de la ampliación y traducción. De esta forma, damos por concluido este segundo ejercicio.

3. Ejercicio 3

3.1. Introducción

En esta práctica se ha implementado un programa en Python para extraer información de la web `https://quotes.toscrape.com/` utilizando el patrón de diseño **Strategy**. El objetivo es obtener las citas, autores y etiquetas de las primeras 5 páginas de la web, utilizando dos estrategias de scraping:

- **BeautifulSoup**: Analiza el HTML de la página directamente con la biblioteca BeautifulSoup.
- **Selenium**: Carga dinámicamente la página y extrae el contenido utilizando un navegador automatizado.

El programa permite elegir entre ambas estrategias, guarda los datos en un archivo YAML y finaliza cuando el usuario lo decide.

3.2. Patrón Strategy e Implementación

El **patrón Strategy** permite definir una familia de algoritmos, encapsularlos y hacerlos intercambiables sin modificar el código del programa principal. En este caso, el scraping se implementa con dos estrategias diferentes:

- **Clase Abstracta**: Define la estructura común a ambas estrategias.
- **BeautifulSoupScraper**: Implementa la extracción de datos con la biblioteca BeautifulSoup.
- **SeleniumScraper**: Utiliza Selenium para cargar dinámicamente las páginas y extraer los datos.
- **ScraperContext**: Administra la estrategia seleccionada y coordina la extracción de las páginas.

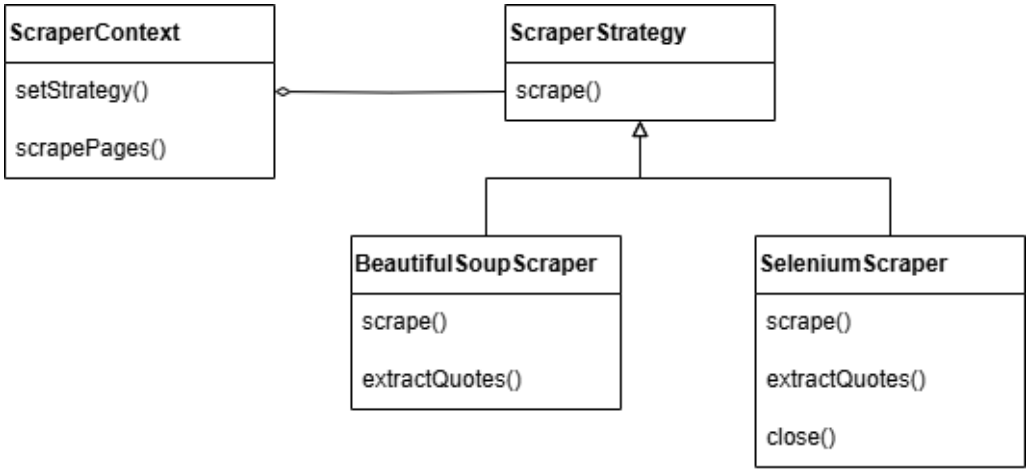

Figura 5: Diagrama Implementado

El usuario puede elegir qué estrategia usar pulsando una tecla (‘B‘ para BeautifulSoup, ‘S‘ para Selenium, ‘Q‘ para salir).

3.3. Dependencias y Funciones

Para la realización de esta práctica se han utilizado las siguientes dependencias:

- **requests**: Para hacer peticiones HTTP a la página web.
- **BeautifulSoup4**: Para analizar y extraer datos del HTML.
- **Selenium**: Para cargar dinámicamente la web y extraer el HTML.
- **PyYAML**: Para guardar los datos extraídos en formato YAML.
- **webdriver-manager**: Para gestionar automáticamente el driver de Selenium.

La información, funciones y recomendaciones necesarias para la realización de este ejercicio han sido obtenidas a través de la documentación oficial de las librerías utilizadas, recursos disponibles en Internet y el apoyo puntual de **ChatGPT** para resolver dudas específicas.

3.4. Funcionamiento del Algoritmo

El algoritmo sigue los siguientes pasos:

1. Muestra un menú para seleccionar la estrategia:

- ‘S‘ → Selenium
- ‘B‘ → BeautifulSoup
- ‘Q‘ → Salir

2. Extrae información de las primeras 5 páginas de `https://quotes.toscrape.com/`:
   - En cada iteración, el programa construye la URL correspondiente a cada página con el formato `https://quotes.toscrape.com/page/{n}/`, donde `n` va del 1 al 5.
   - Si se ha seleccionado la estrategia **BeautifulSoup**, se utiliza la función `requests.get()` para descargar el HTML de la página como texto. Luego, se usa `BeautifulSoup` para parsear el HTML y acceder a los elementos necesarios mediante selectores CSS (como clases).
   - En cambio, si se selecciona la estrategia **Selenium**, se lanza un navegador (de forma oculta usando modo headless) y se accede a la página cargándola completamente, como lo haría un usuario. Se accede directamente al árbol DOM de la página usando funciones como `find_elements(By.CLASS_NAME, ***)` para ir localizando los elementos correspondientes.
   - Cada cita se guarda como un diccionario con tres claves: `text`, `author` y `tags`. Todos los diccionarios se almacenan en una lista que representa todas las citas extraídas.
3. Guarda los datos en un YAML llamado `quotes.yaml`.
4. Imprime en consola las primeras 5 citas extraídas.
5. Vuelve a pedir una opción al usuario hasta que pulse ‘Q‘ para salir.

3.5. Formato de Guardado y Controles del Script

Los datos extraídos se guardan en un archivo llamado `quotes.yaml`, en formato diccionario:

```
- author: "Albert Einstein"
  text: "The world as we have created it is a process of our thinking..."
  tags: ["change", "deep-thoughts", "thinking"]
- author: "J.K. Rowling"
  text: "It is our choices, Harry, that show what we truly are..."
  tags: ["abilities", "choices"]
```

3.6. Conclusión

Tras la implementación y evaluación comparativa de ambas estrategias de scraping, se han podido extraer las siguientes conclusiones:

- La estrategia basada en **BeautifulSoup** presenta un rendimiento significativamente superior en términos de velocidad. Esto se debe a que trabaja directamente sobre el contenido HTML estático obtenido mediante peticiones HTTP, sin necesidad de cargar ni renderizar un navegador.

- Por su parte, la estrategia con **Selenium** ofrece una mayor versatilidad, ya que permite interactuar con páginas que requieren ejecución de JavaScript u otros elementos dinámicos. No obstante, esta flexibilidad conlleva un mayor coste computacional y un tiempo de ejecución más elevado, debido a la inicialización del navegador y la espera a que el contenido se cargue completamente.

En el contexto de esta práctica, dado que la página web analizada no contiene contenido dinámico, la utilización de BeautifulSoup resulta más eficiente y adecuada.

4. Ejercicio 4

4.1. Introducción

En este ejercicio se implementa el patrón de diseño **Intercepting Filter** utilizando el lenguaje de programación Python. Este patrón permite procesar una solicitud antes de que llegue a su destino final a través de una cadena de filtros. Su uso es común en validaciones de datos, autenticación y seguridad.

El objetivo es construir un sistema de autenticación que verifique un correo electrónico y una contraseña antes de permitir el acceso al usuario, utilizando filtros para validar el formato del correo y la seguridad de la contraseña.

4.2. Explicación del patrón

El patrón **Intercepting Filter** se basa en los siguientes componentes:

- **Filtro (Filter):** Define el método `execute`, implementado por cada filtro específico.
- **Cadena de Filtros (FilterChain):** Gestiona la secuencia de filtros y los ejecuta antes de pasar la petición al objetivo.
- **Gestor de Filtros (FilterManager):** Administra la cadena de filtros y facilita su ejecución.
- **Objetivo (Target):** Representa el destino final de la petición una vez que ha pasado por los filtros.
- **Cliente (Client):** Inicia la solicitud, la cual es procesada a través del gestor de filtros antes de llegar al objetivo.

4.3. Explicación de la implementación

Las principales clases y su funcionalidad dentro del sistema son:

- **Filtros:** Validan el correo y la contraseña:
  - `EmailFilter`: Verifica que el correo tenga un formato válido y pertenezca a un dominio permitido.
  - `PasswordLengthFilter`: Comprueba que la contraseña tenga al menos 8 caracteres.
  - `PasswordUppercaseFilter`: Exige al menos una letra mayúscula.
  - `PasswordSpecialCharFilter`: Requiere al menos un carácter especial.

- **Cadena de Filtros:** `FilterChain` mantiene y ejecuta los filtros en orden.
- **Gestor de Filtros:** `FilterManager` encapsula la lógica de los filtros y su ejecución.
- **Cliente:** `AuthenticationService` inicia el proceso de autenticación proporcionando los datos de entrada.

4.4. Resultados obtenidos

El programa solicita al usuario su correo y contraseña, aplicando los filtros definidos. Algunos casos de prueba realizados incluyen:

- **Caso 1:** Correo y contraseña válidos → Autenticación exitosa.
- **Caso 2:** Correo sin "@" → Error: "Correo inválido".
- **Caso 3:** Contraseña sin mayúsculas → Error: "Debe contener al menos una letra mayúscula".
- **Caso 4:** Contraseña sin carácter especial → Error: "Debe contener al menos un carácter especial".

El sistema responde correctamente a distintos escenarios de validación y cumple con el flujo esperado del patrón **Intercepting Filter**.