Desarrollo de Software

Curso 2024-2025

3º - Grado en Ingeniería Informática

Guión de prácticas

Dto. Lenguajes y Sistemas Informáticos

ETSIIT

Universidad de Granada

1 de marzo de 2025

Criterios de evaluación generales

Criterios para superar cada práctica:

- Calidad (se cumplen los requisitos no funcionales)
- Capacidad de trabajar en equipo (reparto equitativo de tareas)
- Implementación completa y verificabilidad (sin errores de ejecución, se cumplen los requisitos funcionales)
- Fidelidad de la implementación al patrón de diseño
- Reutilización de métodos (ausencia de código redundante)
- Validez (de acuerdo con las expectativas del cliente, el profesor en este caso)

Plazos de entrega

Cada práctica completa (código + memoria) será subida a PRADO en una fecha acordada en las clases de prácticas. La nota de cada práctica puede no ser igual para todo el grupo; dependerá del trabajo individual aportado por cada integrante.

- Plazo de Entrega de la P1: 23 de Marzo para todos los grupos.

Práctica 1

Se realizarán programas Orientados a Objetos (OO) bajo distintos patrones de diseño creacionales y estructurales.

Objetivos generales de la práctica 1

1. Familiarizarse con el uso de herramientas que integren las fases de diseño e implementación de código en un marco de Orientación a Objetos (OO)

2. Aprender a aplicar distintos patrones creacionales y estructurales a problemas diversos

3. Adquirir destreza en la práctica de diseño OO

4. Aprender a adaptar los patrones a las especificidades de distintos lenguajes OO

Ejercicio 1. Patrón Factoría Abstracta en Java

Se debe programar utilizando hebras una simulación de dos carreras simultáneas con el mismo número inicial (N) de bicicletas. El valor de N no se conocerá hasta que comience la carrera. Las carreras tienen las siguientes características:

- En la carrera de montaña, se retirará el 20% de las bicicletas antes de terminar.
- En la carrera de carretera, se retirará el 10% de las bicicletas antes de terminar.
- Ambas carreras tienen una duración exacta de 60 segundos.
- Todas las bicicletas se retiran al mismo tiempo.

1 Especificaciones

Se deberán seguir las siguientes especificaciones:

- Se implementará el patrón de diseño **Factoría Abstracta** junto con el patrón **Método Factoría**.
- Se usará Java como lenguaje de programación.
- Se utilizarán hebras para iniciar las carreras de forma simultánea (Se puede usando tanto Thread como Runnable).
- Se implementarán las modalidades de **montaña** y **carretera** como dos familias/estilos de productos.

1.1 Interfaz de Factoría

Se definirá la interfaz Java `FactoriaCarreraYBicicleta` para declarar los siguientes métodos de fabricación públicos:

- `crearCarrera`: Devuelve un objeto de alguna subclase de la clase abstracta `Carrera`.
- `crearBicicleta`: Devuelve un objeto de alguna subclase de la clase abstracta `Bicicleta`.

1.2 Clases y Herencia

- La clase `Carrera` tendrá al menos un atributo `ArrayList<Bicicleta>` con las bicicletas que participan en la carrera.
- La clase `Bicicleta` tendrá al menos un identificador único por carrera.
- Las clases factoría específicas heredarán de `FactoriaCarreraYBicicleta` y se especializarán en un tipo de carrera y bicicleta:
  - `FactoriaMontana`: Implementará los métodos de fabricación para carreras y bicicletas de montaña.
  - `FactoriaCarretera`: Implementará los métodos de fabricación para carreras y bicicletas de carretera.
- Se definirán las clases `Bicicleta` y `Carrera` como clases abstractas o interfaces.
- La factoría de montaña creará los productos `BicicletaMontana` y `CarreraMontana`.
- La factoría de carretera creará los productos `BicicletaCarretera` y `CarreraCarretera`.

Ejercicio 2. Patrón Decorator en Python

Desarrolla un programa en Python que utilice la API de Hugging Face para interactuar con modelos de lenguaje (LLM) y generar resúmenes de texto. Se debe implementar el patrón de diseño *Decorator* para extender la funcionalidad del LLM base, permitiendo añadir dinámicamente funcionalidades adicionales como la traducción del resumen a otro idioma y la ampliación del mismo con detalles adicionales.
Además, el programa deberá configurarse mediante un archivo JSON en el que se especifiquen los siguientes parámetros:

- **texto**: Texto de entrada a resumir.
- **input_lang**: Idioma del texto de entrada.
- **output_lang**: Idioma de salida para la traducción (si se aplica).
- **model_llm**: Modelo LLM (por ejemplo, `"facebook/bart-large-cnn"`) a utilizar para la generación del resumen.
- **model_translation**: Modelo de traducción (por ejemplo, `"Helsinki-NLP/opus-mt-en-es"`) a emplear en la funcionalidad de traducción.
- **model_expansion**: Modelo de generación de texto (por ejemplo, `"facebook/blenderbot-400M-distill"`) a usar para la expansión del resumen.

Se requiere implementar lo siguiente:

1. **Clase base e interfaz `LLM`:** Definir una clase abstracta `LLM` que contenga el método:

   ```
   generate_summary(text, input_lang, output_lang, model)
   ```

   Este método recibirá el texto a resumir, el idioma del texto de entrada, el idioma de salida (para traducción) y el modelo LLM a utilizar.

2. **Clase `BasicLLM`:** Implementar la clase `BasicLLM` que utilice el modelo especificado en el parámetro `model_llm` para generar un resumen del texto de entrada, haciendo uso de la API de Hugging Face sin necesidad de descargar localmente el modelo. (se debe generar un token, gratuito).

3. **Decoradores concretos:** Implementar dos decoradores que extiendan la funcionalidad del LLM base:

   - `TranslationDecorator`: Decora el LLM base para traducir el resumen generado al idioma especificado en `output_lang`, utilizando el modelo definido en `model_translation`.
   - `ExpansionDecorator`: Decora el LLM base para ampliar el resumen con detalles adicionales, empleando el modelo indicado en `model_expansion`.

4. **Código cliente:** El programa principal deberá:

   - Leer la configuración desde un archivo JSON (por ejemplo, `config.json`) con la siguiente estructura:

     ```
     {
       "texto": "Texto de entrada a resumir...",
       "input_lang": "en",
       "output_lang": "es",
       "model_llm": "facebook/bart-large-cnn",
       "model_translation": "Helsinki-NLP/opus-mt-en-es",
       "model_expansion": "facebook/blenderbot-400M-distill"
     }
     ```

   - Instanciar el LLM básico y, opcionalmente, envolverlo en los decoradores `TranslationDecorator` y `ExpansionDecorator` de forma dinámica, según se desee combinar funcionalidades.
   - Imprimir en consola el resumen básico, el resumen traducido, el resumen ampliado, y una combinación de ambos (por ejemplo, primero traducir y luego expandir).

Objetivo: El ejercicio tiene como finalidad que los alumnos aprendan a implementar el patrón *Decorator* para extender funcionalidades de manera dinámica y configurable, integrando llamadas a la API de Hugging Face sin necesidad de descargar modelos localmente, y configurando el comportamiento del programa mediante un archivo JSON.

Ejercicio 3

Implementar un programa en Python que, utilizando el patrón *Strategy*, obtenga información de las 5 primeras páginas del sitio `https://quotes.toscrape.com/`. Se deben extraer los siguientes datos de cada página:

- Texto de la cita.
- Autor.
- Etiquetas asociadas.

La información extraída se debe guardar en un archivo **YAML** en formato diccionario. Se deben implementar dos estrategias para realizar el scraping:

- **BeautifulSoup:** Empleando `requests` y `BeautifulSoup` para obtener y parsear el HTML.
- **Selenium:** Utilizando Selenium para cargar las páginas dinámicamente y luego procesar el HTML.

Ejercicio 4

Un servicio de autenticación requiere validar las credenciales de un usuario (correo y contraseña) sin modificar las clases existentes. Para ello, se debe implementar una solución aplicando el **patrón de filtros de intercepción**. Se deberán crear filtros que intercepten y validen los datos antes de su procesamiento final.

1. Solicitar al usuario que introduzca un correo y una contraseña.
2. Implementar un filtro para el correo que verifique:
   - Que el correo contenga texto antes del carácter `@`.
   - Que el dominio sea `gmail.com` o `hotmail.com`.
3. Implementar tres filtros para la contraseña. **Los filtros para la contraseña serán definidos por el grupo de trabajo y deben contribuir a mejorar la seguridad de la misma**.
4. Cada filtro debe implementarse en una clase separada.
5. Utilizar el patrón de **filtros de intercepción** para interceptar y validar la información.
6. La solución puede implementarse en el lenguaje de programación de su elección

Ejercicio 5. Individual (1 punto)

Un servicio de información geográfica en España proporciona datos en kilómetros y necesita ser compatible con una aplicación de navegación en Los Ángeles, que trabaja con millas. La aplicación de Los Ángeles proporciona rutas con distancias en millas, pero el servicio español necesita estas distancias en kilómetros para integrarlas con su sistema. Objetivo del ejercicio: Implementar una solución que permita al servicio de información geográfica español interpretar y utilizar las distancias provistas por la aplicación de navegación de Los Ángeles. Utilizar un patrón de diseño estudiado. El código debe permitir que el servicio español utilice la información de la aplicación de Los Ángeles sin modificar las clases existentes.

Criterios de puntuación:

- Realización del ejercicio usando el patrón de diseño adecuado. (Hasta 0.8 puntos).
- Uso de sockets, interfaz gráfica (java/python), y/o otras librerías donde se simule el software en un entorno real (Hasta 1.2 puntos, pero se requiere defensa de las librerías externas utilizadas).

Se evaluarán todos los elementos de un diseño de software adecuado.