README - Práctica 02

Datos Generales

Integrantes:
- Aviles Luque Luis Diego - 318176653
- Luis Enrique Flores Juárez - 322278316
- Julio César Islas Espino - 320340594

Materia: Modelado y Programación

Práctica: 02

Anotaciones sobre la Práctica

En esta práctica se implementó un sistema, con el que se controla un robot mesero de una pizzería, dicho robot realiza varias acciones (no simultáneamente) tomar órdenes, preparar helados y pipshas, entregar la orden a los clientes.

La práctica tenía como objetivo implementar los patrones State, Template y Decorator, de modo que el código se puede extender fácilmente y tenga mayor organización.

La práctica cuenta con varias condiciones:

- El robot debe atender a un cliente a la vez.
- Se puede ordenar una pizza, un helado o uno de cada uno.
- La orden puede cancelarse antes de ser confirmada.
- Al confirmar la orden, el robot prepara los productos y entrega los productos.

Argumentación de Diseño (Patrones State, Template y Decorator)

Aquí explicamos cómo los usamos en la práctica.

1. Patrón State

Problema a resolver:

EL robot toma distintos estados (dormido, tomar orden, preparar orden, esperar entrega y entregar). Son varias condiciones a tomar en cuenta para que el robot no realice 2 o más acciones simultáneas.

Aplicación y Justificación:

Se definió la interfaz EstadoRobot y cada estado (EstadoAtendiendo, EstadoDormido, EstadoEntregando, EstadoPreparando) implementa sus respectivos comportamientos y transiciones de un estado a otro. Ejemplo: Después de realizar una entrega el robot regresa a su estado dormido ya que realizó el proceso completo.

Ventaja:

Con esto logramos encapsular los requerimientos para poder cambiar de un estado a otro y mantener un código más limpio, de modo que si se requiere agregar un nuevo estado, solo se necesito agregar otra clase, y las demás no deberán ser modificadas.

2. Patrón Template

Problema a resolver:

Tener código con una "base" ya que las pizzas tienen un patrón similar en su preparación (masa, salsa, queso, horno, empaquetar) aunque se diferencian en el tipo de queso y proteína ocupada posteriormente.

Aplicación y Justificación:

Ocupamos este patrón para crear una base, la cual sigue la secuencia de preparación de pizza, y gracias a esta, podemos tomar diferentes "caminos" con subclases las cuales mantienen el código de las variaciones de los ingredientes de la pizza.

Ventaja:

Evitamos repetir código, ya que la secuencia se centra en una clase la cual es como la "Plantilla" de la preparación de la pizza, o la "base". Después las subclases sobrescribir métodos para las variaciones que lleguen a ordenar los clientes.

3. Patrón Decorator

Problema a resolver:

Mantener un código ordenado y eficaz, ya que los helados pueden llevar varias combinaciones de ingredientes extras, de modo que se necesitan varias combinaciones de clases. .

Aplicación y Justificación:

Con la interfaz Helado, creamos subclases:
HeladoSencillo: Define el sabor y precio de un helado.
DecoratorHelado: Envuelve el helado y agrega un ingrediente extra con el precio actualizado.
ExtraIngredienteHelado: Define que ingredientes extra puede poner al helado y costos.

Ventaja:

Añadir o quitar ingredientes de manera sencilla sin creación de nuevas clases, gracias a la estructura del código, además de que al agregar más ingredientes se sobreescriben métodos y es más ordenado al crear un helado.

Notas adicionales

- El sistema genera una simulación de un robot atendiendo clientes y preparando alimentos.

- Los diagramas UML se entregan en imágenes PNG aparte.

- El código está documentado en formato Javadoc.

- Se respetó el formato de entrega (README en PDF, diagramas en imágenes y código en carpeta src).

Compilación y Ejecución

El proyecto no requiere librerías externas ni configuración especial.

Se puede compilar y ejecutar con los comandos convencionales de Java:

```
javac *.java

java Practica2
```

El proyecto se desarrolló y probó usando:

- Versión de Java: JDK 23

- Sistema operativo: Windows 10 con Git Bash (MINGW64)

- Sistema operativo usado en las pruebas: Windows 10 con Git Bash (MINGW64).

- No se requiere el uso de puertos, IPs, ni elementos de red.

- El programa genera como salida el archivo Practica1.txt con todas las transacciones y mensajes.

- No se utiliza serialización ni características que produzcan warnings adicionales.

Conclusión

En conclusión, implementamos varios patrones para diferentes motivos:
State para estados del robot, Template para crear una base al preparar pizzas y Decorator para las combinaciones de helados y poner ingredientes extra (actualizando también su precio).
Gracias a estos patrones es más sencillo agregar subclases sin modificar las ya existentes, además de mantener orden, y sin existir redundancias en elcódigo..