Práctica 2

Equipo: Los Peaky Blinders
Carlos Emilio Castañón Maldonado José Camilo García Ponce
Dafne Bonilla Reyes

1 **Menciona los principios de diseño esenciales del patrón State, Template e Iterator. Menciona una desventaja de cada patrón.**

❖ **Principios de diseño esenciales:**

- **State**

  Usamos el patrón de diseño State cuando queremos que un objeto cambia su comportamiento en función de un estado interno. Para esto, usamos una variable de estado y un bloque de condición `if-else` y así poder realizar diferentes acciones en función de este estado. State proporciona una forma sistemática y sin acoplamiento para lograrlo a través de implementaciones de contexto y estado.

- **Template**

  El patrón de diseño Template forma parte de la familia de patrones denominados de comportamiento y define un algoritmo como un esqueleto de operaciones, dejando que las clases secundarias implementen los detalles. La estructura general y la secuencia del algoritmo son conservadas por la clase principal.

  Este patrón nace de la necesidad de extender determinados comportamientos dentro de un mismo algoritmo por parte de diferentes entidades. Es decir, diferentes entidades tienen un comportamiento similar, pero que difiere en determinados aspectos puntuales en función de la entidad concreta.

  Template abstrae todo el comportamiento que comparten las entidades en una clase abstracta de la que, posteriormente, extenderán dichas entidades. Esta superclase definirá un método que contendrá el esqueleto de ese algoritmo común y delegará determinada responsabilidad en las clases hijas, mediante uno o varios métodos abstractos que deberán implementar.

- **Iterator**

  Iterator encapsula los detalles del trabajo con una estructura de datos compleja, proporcionando al cliente varios métodos simples para acceder a los elementos de la colección de tal manera que otro código pueda utilizar dichos elementos haciendo posible recorrer cada elemento de la colección sin acceder a los mismos elementos una y otra vez.
  Esta solución, además de ser muy conveniente para el cliente, también protege la colección frente a acciones descuidadas o maliciosas que el cliente podría realizar si trabajara con la colección directamente. Además de que es posible implementar diferentes iteradores que pueden presentar diferentes tipos de recorrido sobre la estructura.

  La idea central del patrón Iterator es extraer el comportamiento de recorrido de una colección y colocarlo en un objeto independiente llamado iterador. Todos los iteradores deben implementar la misma interfaz para que el código cliente sea compatible con cualquier tipo de colección o cualquier algoritmo de recorrido, siempre y cuando exista un iterador adecuado.

❖ **Desventajas:**

- **State**

  El patrón State requiere mucho código para ser escrito. Según la cantidad de métodos de transición de estado diferentes que se definan y la cantidad de estados posibles en los que puede estar un objeto, rápidamente puede haber docenas o más métodos diferentes que deben escribirse. Si hay varios estados y/o muchos objetos, cada uno con su propio estado, necesitamos una clase para cada estado, y el programa general puede volverse fácilmente demasiado complejo, sin contar que el uso de memoria puede ser demasiado alto.

- **Template**

  Algunos clientes pueden verse limitados por el esqueleto proporcionado de un algoritmo y podemos terminar implementando un método que no debería implementarse o no implementar un método abstracto en absoluto. Los métodos Template tienden a ser más difíciles de mantener cuantos más pasos tengan. Además, una vez que la estructura del algoritmo está establecido y tenemos algunas subclases funcionando, es muy complicado modificar el algoritmo, ya que cualquier cambio nos obligaría a modificar todas las subclases.

- **Iterator**

  Las implementaciones típicas de Iterator tienen un comportamiento no definido si la colección se modifica mientras se está iterando, además de que para recorridos simples (como arreglos o listas ordenadas), iterar a través de iteradores pude llegar a ser tedioso y poco eficiente.

2 Uso del programa

– Compilar desde `src/`

```
javac *.java Dishes/*.java Menus/*.java Robots/*.java
```

– Ejecutar desde `src/`

```
java Practica2 <nombreCliente>
```

– Generar la documentación desde `src/`

```
javadoc -d docs *.java Dishes/*.java Menus/*.java Robots/*.java
```

➢ **Explicación:**

Para iniciar el programa, primero es necesario compilar y ejecutar el programa escribiendo el nombre del cliente después de `Practica2/`. Después, los pasos para ordenar un platillo son los siguientes:

1. Se activa al robot.
2. Se pone a caminar al robot.
3. Se pide que se atienda al cliente.
4. Si es necesario se pide el menú.
5. Se pide que se atienda para poder ordenar.
6. Se manda a cocinar al robot.
7. Se pone a cocinar al robot.
8. Se pide que se entregue la comida

Si se quiere generar la documentación, esto sería con el comando dado arriba, y luego, los archivos se generarán en el directorio llamado *docs*.

3 Implementación

Para poder facilitar la forma en la que se muestra el menú al cliente, elegimos usar el patrón de Iterator.

Después, en la parte de las hamburguesas usamos el patrón Template para así poder hacer que ciertas hamburguesas hagan ciertos pasos (principalmente cocinar la carne) de la mejor manera.

Por último, usamos State para que el robot tenga la capacidad de poder hacer las mismas acciones, pero de maneras diferentes dependiendo del punto de la ejecución en el que se encuentre.

Además, para la parte del menú de opciones, intentamos ponerlas en un orden lógico, sin embargo, hay una en particular que quedó extraña: mostrar menú.