Práctica 02

Modelado y Programación

Equipo: El Gallinero

Integrantes:

- Arriaga Santana Estela Monserrat (319044357)
- García Díaz José David (422109033)
- Linares Rios Jatziri (319138335)

Objetivo: *El objetivo de esta práctica es implementar adecuadamente los patrones State, Template e Iterator en la resolución del problema propuesto. La solución propuesta debe evitar caer en problemas de rigidez, fragilidad, inmovilidad, y viscosidad.*

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

STATE

- Principios de diseño:
    - Principio de responsabilidad única: El programa se encarga de organizar el código relacionado con estados particulares en clases separadas.
    - Principio de abierto/cerrado: Se pueden introducir nuevos estados sin cambiar clases de estado existentes o la clase contexto.
- Desventaja:
    - Aplicar el patrón puede resultar excesivo si una máquina de estados sólo tiene unos pocos estados o raramente cambia.

TEMPLATE

- Principios de diseño:
    - Principio de Hollywood: El flujo de ejecución de un programa es completamente distinto o “invertido” a los métodos de desarrollo tradicionales. En lugar de procesar los datos que hay en el momento del proceso, el propio software se encarga de conectar con la entidad correspondiente y obtener el dato que necesita en cada momento.

- Desventaja:
    - Los métodos template tienden a ser más difíciles de mantener cuantos más pasos tengan

ITERATOR

- Principios de diseño:
    - Principio de responsabilidad única: Es posible limpiar el código cliente y las colecciones extrayendo algoritmos de recorrido voluminosos y colocarlos en clases independientes.
    - Principio de abierto/cerrado. Es posible implementar nuevos tipos de colecciones e iteradores y pasarlos al código existente sin descomponer nada.

- Desventaja:
    - Utilizar un iterador puede ser menos eficiente que recorrer directamente los elementos de algunas colecciones especializadas

2. Notas sobre la implementación de la práctica.
   Para la implementación de cada patrón tuvimos las siguientes consideraciones:
    - Template.
      Notamos que el patrón template era el que correspondía a la parte de las hamburguesas y su preparación porque el algoritmo para preparar una hamburguesa es siempre el mismo salvo un par de pasos: el de preparar carne y el de añadir el queso. De manera que el método de "preparaHamburguesa" podemos implementarlo como una serie de pasos en el que la implementación de dos de ellos se deja a las clases de las Hamburguesas específicas.
      Hicimos la distinción entre una hamburguesa y un platillo pensando en que el código fuera flexible por si el restaurante eventualmente quisiera agregar platillos que no son hamburguesas. Dicho esto nos pareció adecuado que las hamburguesas extendieran la clase platillo, porque a fin de cuentas las hamburguesas son platillos del restaurante.
      Dado que la clase platillo contiene un atributo que nos dice si la hamburguesa tiene queso o no, pudimos generalizar el paso de agregar el queso con un condicional dentro del método “preparahamburguesa”, en vez de dejar la implementación a clases específicas. Por lo que finalmente el paso de preparar la carne fue el único que se dejo la implementación a las clases hijas.
    - Iterator.
      Como el Robot debe ser capaz de “mostrar de corrido” los menús, pero cada menú guarda los platillos en una estructura de datos distinta entonces es claro que esta parte de la práctica corresponde a la implementación del patrón Iterator, pues el robot no debe de lidiar con la forma especifica de iterar cada menú según su implementación.
      Para la implementación utilizamos la interfaz Iterator<T> de Java, de manera concreta nos interesó tener acceso a los platillos de cada menú a través de iteradores de tipo Iterator<Platillo>.
      Decidimos usar una clase para cada uno de los tres tipos de menús y añadirlos como atributos en una clase llamada MenuCompleto, esto con el objetivo de poder acceder a todos los menús con una sola clase, pero a la

vez dejando la posibilidad de que cada menú tenga métodos y atributos específicos que se pudiesen requerir en un futuro.
Implementamos iteradores para el arreglo del menu general y la tabla hash del menu especial, ya que estas estructuras de datos no tienen un método para regresar iteradores directamente, mientras que el array list del menu diario sí. Los iteradores que hicimos de nuestra propia mano desde luego deben implementar la interfaz Iterator<Platillo>.
Con esta implementación el robot puede acceder a un iterador de cada menú a través de una instancia de MenuCompleto y puede leer de corrido el menú que él desee.

- State.
  El robot puede realizar o no realizar acciones dependiendo de lo que esté haciendo, por lo que claramente esta parte de la práctica corresponde a implementar el patrón state.
  Para este patrón identificamos cuatro estados y siete acciones. El estado atendiendo, resultando el más “complejo” pues cuando el robot atiende debe de poder realizar las acciones de leer el menú y tomar el pedido del usuario para ponerse a cocinar. La acción de leer el menú requiere como parámetro un menú completo para que el robot pueda obtener los iteradores de los menús de algún lado y leer la información de los platillos, ya que según indicaciones de la práctica no los puede tener como atributos. Tuvimos que considerar el caso en que el usuario pide algo en el menú que no existe en cuyo caso el robot deberá seguir atendiendo. Además las acciones para tomar una orden y cocinar requieren de dos parámetros, una instancia del menú completo y el identificador del platillo seleccionado ya que decidimos tomar la entrada del usuario desde el main y había que pasar el platillo seleccionado al robot. El resto de las acciones: activar, suspender, caminar y entregar comida fueron más simples de implementar pues más bien establecen transiciones entre algún estado y otro.

Comentarios adicionales:

-Para correr el programa:

Para ejecutar nuestro programa primero tendrá que situarse en la carpeta src, para después compilar la clase principal y poner en la terminal "javac Main.java", una vez compilado ahora se ejecutará con "java Main".