Proyecto 1

Modelado y Programación

Vaquitas computitas

- 321088642 Flores Morán Julieta Melina
- 321244763 López Pérez Mariana
- 321102502 Zarco Romero José Antonio

Implementación de cada patrón

Comportamiento

Strategy

Con el objetivo de que la tienda CheemsMart brinde una mejor experiencia de usuario diseñamos una interfaz común **“**`ManejadorDelIdioma.java`**”**, para encapsular los mensajes de texto interactivos que la tienda tiene con el usuario. Siguiendo el principio de diseño de encapsular lo que cambia en tiempo de ejecución es que tomamos la decisión de definir esta familia de algoritmos, encapsularlos y hacerlos intercambiables para que el idioma difiera según quien inicia sesión. De esta manera, creamos clases concretas, tales como **“**`ManejadorDelIdiomaLatino.java`**”**, “`ManejadorDelIdiomaEspanol.java`”, “`ManejadorDelIdiomaEstadounidense.java`**”**, para manejar por separado los mensajes relacionados con el país del cliente. Así, la interfaz de `CheemsMart` puede intercambiar dinámicamente el idioma, con base en la información del cliente.

Iterator

La razón de emplear iterator, fue encapsular la información del catálogo de la tienda CheemsMart, con lo cual el controlador de la interfaz principal, “`ControladorCheemsMart.java`**”**, tenga acceso de solo lectura a los productos de cada departamento, pues únicamente guarda el catálogo como un iterable; así tenemos una forma de acceder a los productos del catálogo de manera secuencial, sin exponer la representación

interna del mismo. En consecuencia, no solo abstraemos la representación subyacente del catálogo, también ganamos simplicidad a la hora de recorrer los elementos.

Estructurales

○ *Proxy*

Empleamos un Proxy de protección para controlar el acceso a los objetos relacionados con la información sensible de la tienda , como la base de datos del cliente y el catálogo original. Gracias a la interfaz "`Tienda.java`", tenemos la tienda con la información sensible, pero proporcionamos al controlador de la interfaz del usuario un sustituto que controla el acceso a la tienda original, la tienda Proxy actúa como un intermediario entre el controlador de la interfaz del cliente y el objeto real de la tienda; proporcionando un nivel adicional de control y seguridad. Además de la información de la tienda, también es importante cuidar la información sensible del cliente, como lo es su cuenta bancaria y su información personal, por esto la interfaz "`Cliente.java`" permite que "`ClienteProxy.java`" sea intermediario entre el objeto que tiene la información del cliente real y el controlador de la interfaz del usuario, con el fin de no darle pleno acceso a esta información al usuario que inicia sesión.

○ *Composite*

Utilizamos el patrón Composite para representar la estructura jerárquica de los departamentos y productos en el catálogo de la tienda CheemsMart, por medio de la clase abstracta "`CatalogoComponente.java`". Esto nos permite tratar tanto a los productos individuales como a los departamentos de productos de manera uniforme, facilitando la manipulación y la iteración de la estructura del catálogo, así como también volviendo fácil agregar más productos, departamentos e incluso sub departamentos si en un futuro se quiere extender el catálogo.

○ *Decorator*

Respecto a los descuentos, decidimos usar el patrón Decorator, para separar a los productos de la lógica de los descuentos, siguiendo así el principio de responsabilidad única. Además, usar Decorator permite aplicar descuentos de manera dinámica a los productos en tiempo de ejecución, ya que, cuando generamos la oferta, aplicamos el descuento solamente a los productos del departamento de preferencia del cliente. También, proporcionamos una manera transparente de reutilizar fácilmente la lógica de descuento en diferentes contextos y para diferentes productos, pudiendo incluso acumularlos si se llega a permitir tener descuentos adicionales. Así, podemos adjuntar esta característica adicional a los productos de forma dinámica.

Creacionales

- ### *Prototype*

En este proyecto, utilizamos Prototype para tener una copia del catálogo individual para cada cliente, el patrón nos permitió poder hacer nuevas instancias del catálogo copiando la instancia existente que guarda la tienda. Esto con dos propósitos, el primero es asegurar la integridad del catálogo original al no permitirle al usuario acceder a él, el segundo es que de esta manera, le podemos mostrar un catálogo donde el descuento, si le ha correspondido, ya se ha aplicado y así personalizar los precios.

La clase "`CatalogoComponente.java`" implementa la interfaz "`Cloneable`", y bien podríamos usar el método 'clone()', sin embargo, la clonación no se realizaba de manera profunda correctamente, por lo que escribimos un método 'copia()' para clonar componentes del catálogo de manera profunda, creando nuevas referencias independientes que son a las que tiene acceso el controlador de la interfaz del usuario.

- ### *Singleton*

También, empleamos el patrón Singleton para garantizar que solo haya una instancia de ciertas clases importantes en el sistema, como la "`BaseDeDatosClientes.java`" y la tienda real "`CheemsMart.java`". Evitando así problemas de concurrencia, y manteniendo un mayor control al crear dichos recursos, cuyo coste de creación es significativo para el rendimiento del programa, ya que al momento de escalar a red, pueden generarse problemas al mantener varias conexiones abiertas y queremos que siempre se esté operando sobre la misma instancia de la base y de la tienda para evitar pérdida de información. Además, mantiene coherencia en la gestión del inventario y todas sus operaciones relacionadas.

- ### *Simple Factory*

Si bien puede debatirse que Simple Factory sea un modismo de programación, implementamos dicho patrón en la clase "`FabricaSimplePaises.java`", se puede considerar una fábrica simple, ya que tiene métodos exclusivos para crear instancias de manejadores del idioma y catálogos con descuento basados en el país de origen del cliente de manera encapsulada e independiente al código principal. Manteniendo el código principal limpio y fácil de entender, ya que el proceso de creación de instancias se maneja en un solo lugar.

Modelo-Vista-Controlador

El modelo está representado por la gran mayoría de las clases, pues estas gestionan la lógica de negocio y los datos de la tienda, entre ellas se encuentran "`Tienda.java`", "`Carrito.java`", "`Cliente.java`", entre otras. Mientras que la vista es manejada por la clase "`InterfazCheemsMart.java`" e "`InicioSesion.java`", encargadas de mostrar información al usuario. Los controladores, representado por

“ControladorCheemsMart.java” y “ControladorInicioSesion.java” actúan como intermediarios entre el modelo y la vista, gestionando las interacciones del usuario y comunicándose con el modelo para realizar las diferentes consultas y acciones. Así tenemos organizado independientemente la forma visual en que se presenta al usuario el programa, la información que maneja el programa y cómo interactúan.

Comentarios

Manejar el Dinero del Usuario

Cada cliente tiene asociado a él un objeto cuenta bancaria, que, análogamente al caso de la vida real, almacena y maneja las operaciones relacionadas con el dinero del usuario y tiene un identificador que es el número de cuenta. El dinero de los usuarios es un atributo que se genera al mismo tiempo que se crea al cliente, pero puede cambiar en tiempo de ejecución, en particular para esta práctica solo necesitamos que disminuya. Al momento de realizar una compra, la tienda se encarga de cargar el monto al cliente, disminuyendo la cantidad en su cuenta bancaria, en caso de que el dinero en esta no sea suficiente, se manda un mensaje al respecto y se cancela la operación.

Descuentos

De acuerdo con los requerimientos, los descuentos están segmentados por países. Siendo que la cantidad de descuento aplicada y a qué departamento se aplica debe ser igual para todos los clientes de una región.

Decidimos no usar Observer, ya que esta es una característica que no es por cliente y que no es necesario avisarles a menos que estén directamente en el menú principal, donde es el único punto que realmente se les informa de su descuento en el programa y donde realmente necesitan saberlo, ya que si no está el cliente en el sistema probablemente no tenía intención de saber de su descuento.

Decidimos que la base de datos almacene la cantidad de descuento que aleatoriamente un método genera para cada país, así, cada que un usuario inicie sesión sabemos qué cantidad le corresponde según su región y también se define qué tipo de descuento, es decir, a qué departamento se aplicará.

Así, se pueden cambiar los montos con el método en base de datos dinámicamente de ser necesario, aunque para este proyecto no lo fue y tener la información guardada para aplicarlo según que usuario inicie sesión.

Vista Compra Segura

Las razones que nos llevaron a darle múltiples oportunidades al cliente de ingresar sus datos bancarios cuando falla la verificación de credenciales fueron justamente mejorar la experiencia de usuario, pues esta acción reduce la posibilidad de abandono de la aplicación debido a errores simples, como lo son tipográficos, que generen frustración en el cliente.

Desechamos la opción de cerrar el sistema de inmediato después de un solo error, ya que este podría ser percibido como una acción brusca y como usuarios sabemos que es muy común cometer ligeros erraros al escribir, sobre todo un número de cuenta que es una serie de caracteres aleatorios. Además, podría potencialmente exponer al usuario a riesgos de seguridad si otros intentan acceder a su cuenta después de un cierre repentino.

Salir del Sistema

Incluimos la opción de salir del sistema en el menú de inicio de sesión de la terminal, pues consideramos mejora la experiencia del usuario al proporcionar una forma intuitiva y consistente para salir del sistema. Además, reduce las posibilidades de problemas como la pérdida de datos no guardados o la corrupción del sistema en un incorrecto cierre del programa. Así, el usuario tiene que cerrar su sesión para posteriormente salir del sistema, lo cual puede hacer desde la mayoría de momentos en el programa, y consideramos que es bastante amigable e intuitivo para el usuario

Nota del UML: El editor de diagramas dia no nos permitió ajustar todos los recuadros y hay algunos pequeños desbordes cuando contienen textos largos.

Nota sobre carpeta: Optamos por crear una carpeta “igu” de interfaz gráfica de usuario para separar los archivos que engloban la parte con la que el usuario tiene acceso directo. Esto para tener mejor modularizado el sistema e identificar mejor que parte corresponde a lo que es gráfico y de interacción con el cliente.

Ejemplos de Entradas

Nombre de Usuario	Contraseña	Nombre	Teléfono	Dirección	Dinero	Número de Cuenta Bancaria	País de Origen	ID
Pancho	1234	Francisco Rodriguez	5528203722	Tlalpan, CDMX	1500	1234567890	MEXICO	1

Luillilol	c0ntr4senA	Luis Perez	5587203732	Venustiano Carranza, CDMX	99999	DFNF93RR	ESTADOS_UNIDOS	2
Lalo	0123	Eduardo	5561443440	Espana	10	DJNFKABFVR9JF	ESPANA	3

Ejecución

Esta práctica fue desarrollada en Fedora 38 y java version 17.0.9.

Utiliza el comando *javac* para compilar los archivos fuente de Java.

Se ejecutaría de la siguiente manera:

```
$ javac src/*.java

$ java src.Proyecto1
```