UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Proyecto #1
Equipo: Evasores de Impuestos
Tienda CheemSmart

PRESENTAN

Castro Hernández Rafael
320051809

Gallegos Cortes José Antonio
320316566

Martínez Leal José María
317243970

PROFESORA

Rosa Victoria Villa Padilla

ASIGNATURA

Modelado y Programación

1 Compilación y Ejecución del Programa

Version java: JDK 17

Para ejecutar nuestro **Proyecto 1** no es necesario instalar ningún otro programa, más que `java`, ya que su ejecución solo será en terminal.

En primer lugar, para compilar el proyecto, se debe acceder desde la terminal a la carpeta `src` donde se encuentran los archivos y escribir el siguiente comando para compilarlos:

```
javac *.java
```

Después de compilarlos con éxito, podemos ejecutar el proyecto partiendo desde nuestro main ejecutable, escribiendo, desde la ubicación del punto anterior:

```
java ZMAINCHEEMS
```

2 Clientes cargados en el sistema

Nuestra tienda virtual ya viene con clientes cargados para evaluar el funcionamiento del programa:

Clientes del sistema			
Usuario	Nacionalidad	Contraseña de Usuario	Clave Bancaria
JuanHorse938	Mexicana	1234	chilepiquin
CdeCiencia	Española	vivaEspana	motomami
Arthur	Estadounidense	333221	capybara

Al inicio de la ejecución se solicita el nombre de usuario y la contraseña de este. Al momento de pagar se pide nuevamente la contraseña del usuario (para mayor seguridad) y la clave de su cuenta bancaria.

3 Patrones de diseño utilizados

A continuación, enumeramos los patrones de diseños usados en este proyecto:

- Strategy, para las vistas por nacionalidad
- Decorator, para el carrito de compras (productos)
- Proxy, para la cuenta bancaria y el catalogo
- Iterator, para mostrar el catálogo

4 Justificación de uso

- **Strategy:** Recordemos que el patrón Strategy nos permite definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables. En nuestro caso, de que necesitamos cambiar la interfaz de la tienda virtual dependiendo del país del cliente, lo más cómodo sería tener diferentes "estrategias de interfaz" implementadas por las clases concretas (VistaMexicano, VistaEspañol, etc.). Esto sería muy útil a futuro ya que facilita el mantenimiento y la extensibilidad del código, ya que nuevas clases concretas que implementen la interfaz definida por el Strategy pueden agregarse en el futuro sin necesidad de modificar el código existente. Además, podemos separar las acciones pues cada estrategia está encapsulada en su propia clase. Por último, elegimos el patrón Strategy ya que nos permite reutilizar código al máximo, ya que la lógica común de la tienda virtual puede permanecer intacta mientras que las estrategias específicas para cada país pueden reutilizar o adaptar partes del código según sea necesario. No elegimos otros patrones de comportamiento similares como Template o State debido a que no ofrecen la misma flexibilidad para intercambiar algoritmos. Por ejemplo, el patrón Template se enfoca en definir la estructura de un algoritmo, lo cual no se adapta adecuadamente a nuestra necesidad de cambio de la interfaz según el país del cliente. Por otro lado, el patrón State se utiliza para cambiar el comportamiento de un objeto en función de su estado interno, lo cual no sería tan adecuado en este caso.

- **Decorator:** Nosotros pensamos que la mejor opción para crear los productos en la tienda virtual es utilizar el patrón Decorator. Este patrón nos permitió agregar funcionalidades adicionales a los objetos de productos de manera dinámica y flexible, sin necesidad de modificar su estructura básica. Con el patrón Decorator, pudimos componer una variedad de productos combinando diferentes características de manera dinámica, lo que hace más flexible la creación de productos. Además, pudimos reutilizar código ya que el patrón permite que diferentes características se implementen como decoradores independientes, lo que mejora la modularidad y la mantenibilidad del código. El patrón también hace que el programa sea escalable ya que permite agregar nuevas funcionalidades a los productos en el futuro mediante la creación de nuevos decoradores. En cuanto a otros patrones estructurales similares como Adapter o Composite, los descartamis porque no ofrecen la misma flexibilidad para agregar funcionalidades adicionales de manera dinámica y modular como lo hace el Decorator. Además, para la creación de productos, preferimos Decorator sobre otros patrones, como Factory o Abstract Factory, ya que estos últimos están más orientados a la creación de familias de objetos relacionados, mientras que el Decorator se centra en agregar funcionalidades adicionales a objetos existentes de manera dinámica, que es como queriamos manejar los productos de nuestra tienda debido a las acciones que

realizamos con ellos.

- **Proxy:** Si pudiéramos describir al patrón Proxy en una palabra sería Seguridad, es por esto que la mejor opción para abordar los requisitos de seguridad y de la representación remota del catálogo de nuestro proyecto es utilizar el patrón Proxy. Notemos que el Proxy actúa como un intermediario entre el cliente y el objeto real, controlando el acceso a este último y proporcionando una capa adicional de seguridad y funcionalidad. En este caso, el Proxy puede manejar la carga remota del catálogo asegurando que la tienda no tenga acceso directo al catálogo real y evitando posibles vulnerabilidades de seguridad. Además, el Proxy garantiza que se cumplan las cotas de seguridad de la tienda ya que funciona como el representante para la acción que supone más riesgo, el pago por medio de una cuenta bancaria. El patrón Proxy fue una elección unánime e instantánea ya que, de todos los patrones vistos en clase (incluyendo a sus hermanos estructurales), es el que más se centra en la seguridad debido a sus características.

- **Iterator:** Elegimos al patrón Iterator como mejor opción para mostrar el catálogo de productos de la tiendita ya que este proporciona una forma eficiente, flexible y elegante de recorrer y acceder a los elementos de una colección sin exponer la estructura interna. Al emplear un Iterator, pudimos iterar sobre el catálogo de productos de manera uniforme y sin importar la implementación subyacente de la colección. Esto nos facilitó el control de nuestro programa debido a que podemos acceder a los elementos de manera secuencial sin necesidad de cargar todo el catálogo en memoria de una sola vez. Además, el Iterator ofrece la posibilidad de implementar iteraciones personalizadas si fuera deseado en el futuro. Todo esto nos permite mostrar el catálogo en una presentación ordenada y controlada de los productos en la tienda virtual. Además el uso de este patrón nos ayuda a mantener otros elementos deseables como la encapsulación y mantenibilidad. Nosotros elegimos este patrón sobre otros, también de comportamiento como Strategy, ya que, aunque ofrecen flexibilidad y otros elementos deseables, ninguno aborda directamente la problemática de la iteración y acceso a elementos de una colección como lo hace el patrón Iterator.