Claro que sí, con mucho gusto. Aquí está la información sobre el examen, completamente en español.

No te preocupes, puedo ayudarte con esto. La prueba te pide que crees un programa en Java para un sistema de pedidos en línea de una tienda de ropa, aplicando patrones de diseño y arquitectónicos específicos. El objetivo principal es crear un sistema de descuentos flexible. Analizaremos los pasos uno por uno para asegurarnos de que comprendas los conceptos y puedas completar la tarea.

**Entendiendo los Conceptos Clave**

Primero, desglosemos los conceptos principales que debes aplicar: MVC, Singleton, Decorator y Command.

**MVC (Modelo-Vista-Controlador)**

El

**patrón arquitectónico MVC** es una forma de organizar tu código en tres componentes interconectados para separar los datos (**Modelo**), la interfaz de usuario (**Vista**) y la lógica (**Controlador**).

* **Modelo**: Esta capa maneja los datos y la lógica del negocio. Es el núcleo de la aplicación e independiente de la interfaz de usuario. Para este proyecto, debes crear al menos tres modelos para

**productos**, **pedidos** y **usuarios**. La clase

DiscountManager también debe estar aquí.

* **Vista**: Esta es la capa de la interfaz de usuario que muestra los datos del Modelo al usuario. Debes desarrollar al menos tres interfaces de usuario que permitan mostrar productos, carritos de compra y aplicar descuentos.
* **Controlador**: Esta capa actúa como intermediario, manejando la interacción del usuario desde la Vista y actualizando el Modelo en consecuencia. Tendrás que implementar al menos tres controladores para manejar la interacción entre la vista y el modelo, incluyendo la aplicación de descuentos.

**Patrón Singleton**

El

**patrón Singleton** asegura que una clase tenga **una sola instancia** y proporciona un punto de acceso global a ella. Utilizarás esto para tu clase

DiscountManager para que todas las partes del sistema usen la misma lógica de descuento. Esto se logra haciendo que el constructor de la clase sea privado y proporcionando un método estático

getInstance() para obtener la única instancia de la clase.

**Patrón Decorator**

El

**patrón Decorator** te permite agregar nuevas funcionalidades a un objeto de forma dinámica sin alterar su estructura. Utilizarás este patrón para aplicar diferentes tipos de descuentos de manera flexible.

* Tendrás que definir la interfaz

Component, estableciendo métodos comunes para los componentes.

* Tendrás que implementar una clase concreta que represente los productos y su lógica de descuento base, la cual debe implementar 'Component'.
* Tendrás que implementar una clase abstracta

Decorator, que extienda 'Component' y tenga una referencia a un objeto 'Component'.

* Por último, tendrás que desarrollar al menos dos clases concretas que extiendan 'Decorator' para agregar funcionalidades específicas, como diferentes tipos de descuentos o promociones.

**Patrón Command**

El

**patrón Command** encapsula una solicitud como un objeto, lo que te permite parametrizar clientes con diferentes solicitudes, ponerlas en cola o registrarlas. En tu caso, lo usarás para acciones como agregar o eliminar productos del carrito.

* Debes crear una interfaz llamada

Command que defina un método Ejecutar.

* Debes implementar al menos dos clases concretas que implementen la interfaz

Command y contengan la lógica de descuento específica.

**Guía de Implementación Paso a Paso**

Ahora que hemos cubierto los conceptos, aquí están los pasos a seguir para crear tu solución, basados en las instrucciones del examen:

**1. Configura tu Entorno**

Primero, asegúrate de tener

**Apache NetBeans** (se recomienda la versión 18) y el **JDK** instalados. Puedes encontrar los enlaces de descarga en las instrucciones.

**2. Implementa la Estructura MVC**

Comienza estructurando tu proyecto en las tres carpetas principales:

Model, View y Controller.

* **Capa del Modelo**: Crea al menos tres clases de modelo para Producto, Pedido y Usuario.
* **Singleton para DiscountManager**: Dentro de la carpeta Model, crea la clase DiscountManager. Haz que su constructor sea privado y agrega un método estático

getInstance() para asegurar que sea un verdadero Singleton.

* **Capa de la Vista**: Crea al menos tres clases de interfaz de usuario (por ejemplo, ProductView, ShoppingCartView, DiscountView).
* **Capa del Controlador**: Implementa al menos tres clases de controlador (por ejemplo, ProductController, OrderController, DiscountController) que manejarán la lógica y las interacciones entre tus modelos y vistas.

**3. Implementa el Patrón Decorator**

Concéntrate en la lógica de descuento dentro de tu clase

DiscountManager.

* Crea una interfaz

Component.

* Crea una clase concreta, por ejemplo, BasicProduct, que implemente Component. Esto representará un producto antes de que se aplique cualquier descuento.
* Crea una clase abstracta

Decorator que extienda Component.

* Crea al menos dos clases

Decorator concretas, como PercentageDiscountDecorator y CategoryDiscountDecorator, para aplicar diferentes reglas de descuento.

**4. Implementa el Patrón Command**

Este patrón manejará las acciones de tu sistema.

* Crea una interfaz

Command con un único método, Ejecutar().

* Crea al menos dos clases concretas que implementen la interfaz Command, como AddProductCommand y RemoveProductCommand. Estas clases encapsularán la lógica para sus respectivas acciones.