Práctico 7: Interfaces y Excepciones

Objetivo:

Desarrollar competencias avanzadas en Programación Orientada a Objetos (POO) en Java, comprendiendo y aplicando la herencia múltiple, el uso de interfaces y el manejo de excepciones para diseñar código flexible, extensible y robusto.

Resultados de aprendizaje:

1. **Diseñar e implementar modelos basados en interfaces en Java**: El estudiante será capaz de crear e implementar interfaces para definir comportamientos comunes y resolver problemas prácticos mediante la integración de interfaces y herencia.
2.
3. **Manejar excepciones en Java para asegurar la robustez del software**: El estudiante será capaz de identificar y gestionar excepciones comunes en aplicaciones Java, utilizando estructuras de control de errores como `try-catch-finally` y aplicando `multicatch` para manejar múltiples excepciones de manera eficiente.
4.
5. **Desarrollar técnicas avanzadas de manejo de errores y recursos en Java**: El estudiante será capaz de lanzar y gestionar excepciones personalizadas con `throw` y `throws`, aplicar las diferencias entre excepciones **checked** y **unchecked**, y utilizar `finally` y `try-with-resources` para liberar recursos de manera segura y eficiente.

En la Parte 1, el alumno debe resolver las katas sobre interfaces. En la parte 2, resolver ejercicios sobre excepciones.

Parte 1 Katas sobre Interfaces

¿Qué es una Kata y cómo se utiliza en programación?

Una kata es un ejercicio de programación diseñado para mejorar habilidades de codificación mediante la repetición y el aprendizaje progresivo. El término proviene de las artes marciales, donde las katas son secuencias de movimientos que se practican repetidamente para perfeccionar la técnica.

En programación, las katas ayudan a los programadores a reforzar conceptos, mejorar la comprensión del código y desarrollar buenas prácticas. **Se recomienda resolver una kata varias veces, intentando mejorar el código en cada iteración, utilizando mejores estructuras, nombres más claros y principios de diseño.**

Importante:

Intentar resolver cada kata sin mirar la solución.

Comprobar la solución y corregir errores si es necesario.

Repetir 2 o 3 veces para mejorar la comprensión, lógica y el código.

Experimentar con diferentes valores para reforzar el aprendizaje.

Resolver Katas

A continuación, te presento 3 enunciados de katas en Java para fortalecer los conceptos vistos en el módulo 7.

Estas katas permiten practicar desde conceptos básicos de interfaces hasta la resolución de problemas complejos como el problema del diamante, aplicando los conceptos vistos en el módulo 7.

Parte 1 Katas de Programación en Java: Uso de Interfaces en un E-commerce

Estas **katas** integran el uso de **interfaces** en el contexto del **Procesamiento de Pedidos en un E-commerce**, siguiendo el contenido del módulo 7 sobre herencia múltiple, interfaces y su aplicación en Java.

Kata 1.1: Baja Complejidad - Uso Básico de Interfaces

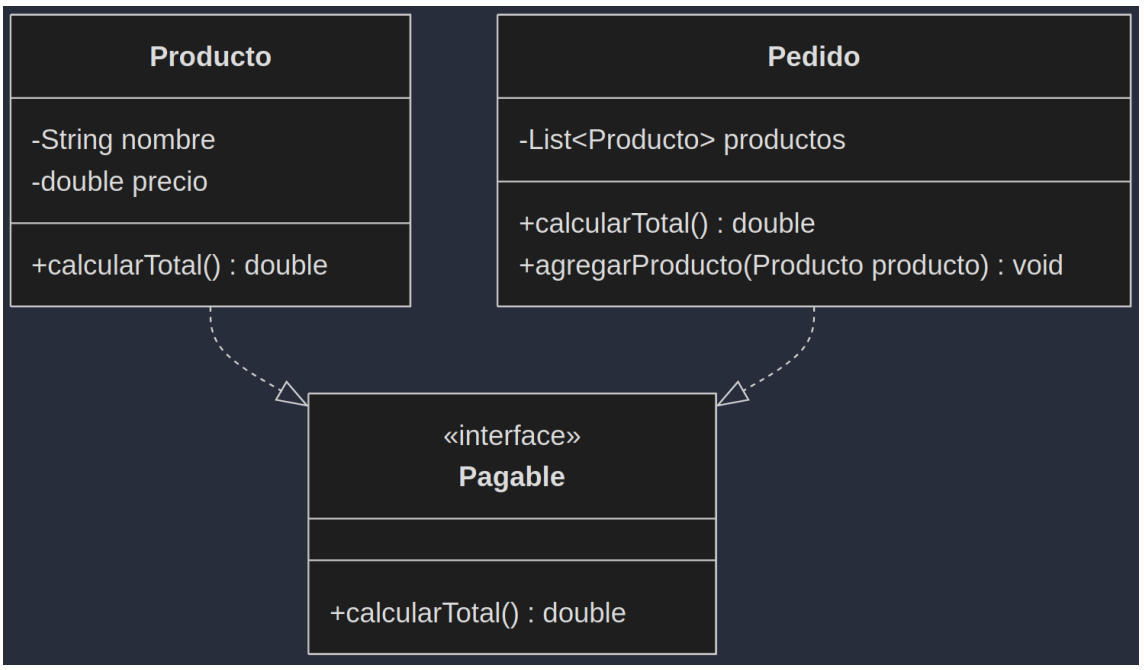
Enunciado

Crea un sistema básico para gestionar pedidos en un e-commerce. Define una interfaz `Pagable` que represente la capacidad de un objeto de ser pagado. Implementa esta interfaz en las clases `Producto` y `Pedido`, y agrega un método para calcular el total a pagar.

Clases

1. **Interfaz `Pagable`**
    - **Métodos:**
        - `calcularTotal()`: Devuelve el total a pagar.
2. **Clase `Producto`**
    - **Atributos:**
        - `nombre`: Nombre del producto.
        - `precio`: Precio del producto.
    - **Métodos:**
        - `calcularTotal()`: Devuelve el precio del producto.
3. **Clase `Pedido`**
    - **Atributos:**
        - `productos`: Lista de productos en el pedido.
    - **Métodos:**
        - `calcularTotal()`: Devuelve la suma de los precios de todos los productos en el pedido.

Diagrama de Clases



🎯 Tarea a realizar

- Implementar la interfaz `Pagable` y las clases `Producto` y `Pedido`.
- Crear un objeto de tipo `Pedido`, agregar varios productos y calcular el total a pagar.

Kata 1.2: Media Complejidad - Herencia entre Interfaces

Enunciado

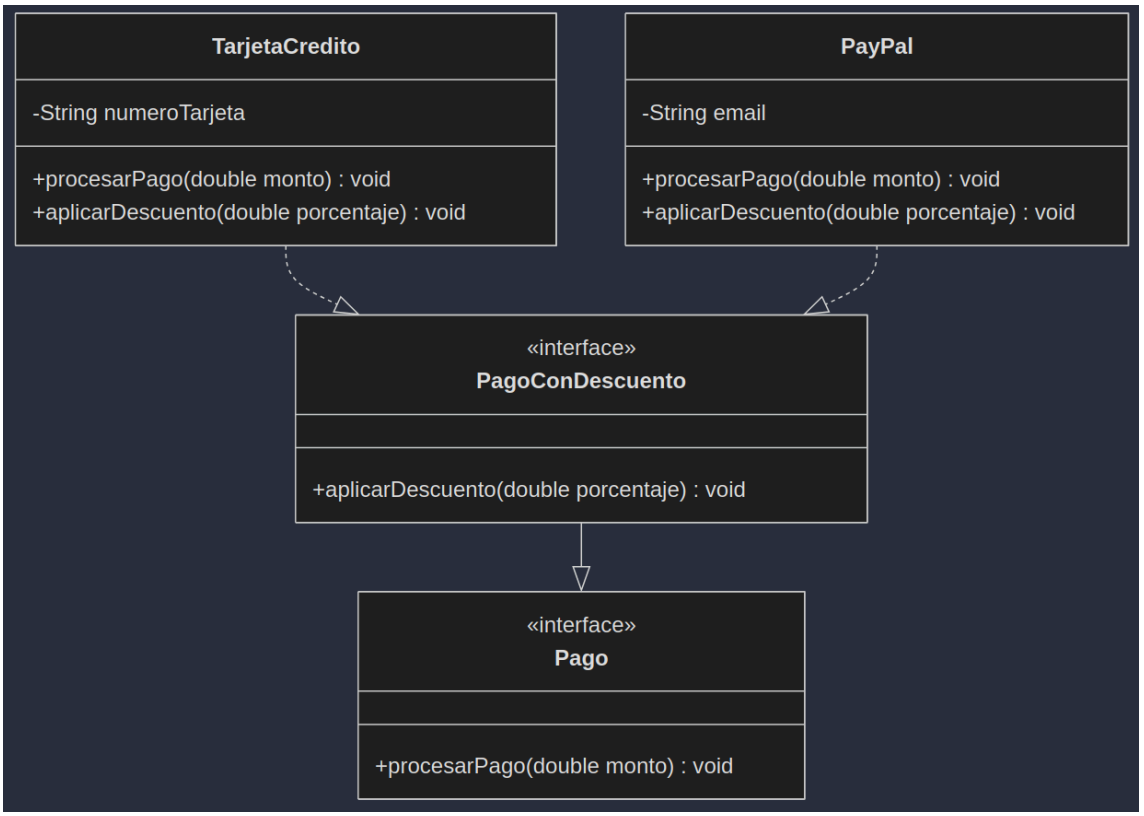
Amplía el sistema de pedidos para incluir diferentes tipos de pagos. Define una interfaz `Pago` que represente la capacidad de procesar un pago. Luego, crea una interfaz `PagoConDescuento` que herede de `Pago` y agregue un método para aplicar descuentos. Implementa estas interfaces en las clases `TarjetaCredito` y `PayPal`.

Clases

1. **Interfaz `Pago`**
    - **Métodos:**
        - `procesarPago(double monto)`: Procesa el pago de un monto específico.
2. **Interfaz `PagoConDescuento`**
    - **Métodos:**
        - `aplicarDescuento(double porcentaje)`: Aplica un descuento al monto del pago.
3. **Clase `TarjetaCredito`**
    - **Atributos:**
        - `numeroTarjeta`: Número de la tarjeta de crédito.
    - **Métodos:**
        - `procesarPago(double monto)`: Procesa el pago con tarjeta de crédito.
        - `aplicarDescuento(double porcentaje)`: Aplica un descuento al monto.
4. **Clase `PayPal`**
    - **Atributos:**
        - `email`: Correo electrónico asociado a PayPal.
    - **Métodos:**
        - `procesarPago(double monto)`: Procesa el pago con PayPal.

- `aplicarDescuento(double porcentaje)`: Aplica un descuento al monto.

Diagrama de Clases



🎯 Tarea a realizar

- Implementar las interfaces `Pago` y `PagoConDescuento`.
- Crear objetos de tipo `TarjetaCredito` y `PayPal`, procesar pagos y aplicar descuentos.

Kata 1.3: Alta Complejidad - Resolución del Problema del Diamante con Interfaces

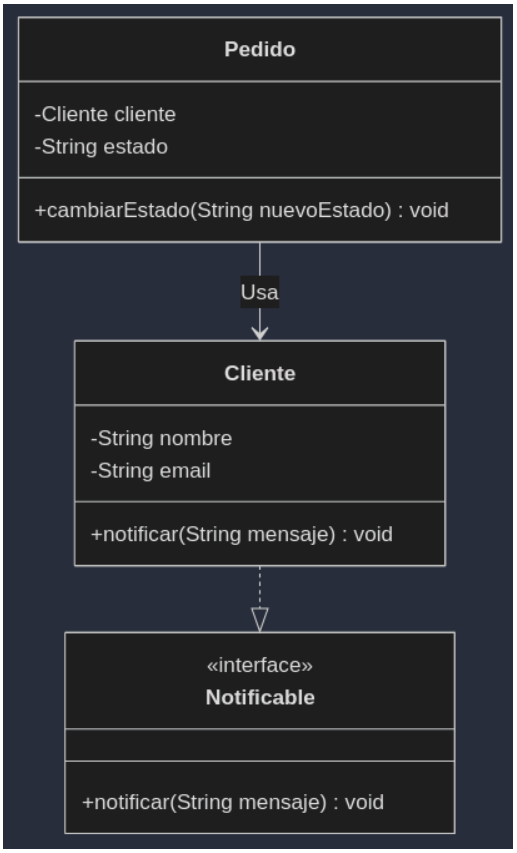
Enunciado

Extiende el sistema de pedidos para incluir la capacidad de notificar al cliente sobre el estado del pedido. Define una interfaz `Notificable` con un método `notificar(String mensaje)`. Luego, crea una clase `Cliente` que implemente `Notificable` y una clase `Pedido` que use esta interfaz para notificar al cliente cuando el estado del pedido cambie.

Clases

1. **Interfaz `Notificable`**
    - **Métodos:**
        - `notificar(String mensaje)`: Notifica al cliente con un mensaje.
2. **Clase `Cliente`**
    - **Atributos:**
        - `nombre`: Nombre del cliente.
        - `email`: Correo electrónico del cliente.
    - **Métodos:**
        - `notificar(String mensaje)`: Envía una notificación al cliente.
3. **Clase `Pedido`**
    - **Atributos:**
        - `cliente`: Cliente que realizó el pedido.
        - `estado`: Estado actual del pedido (Pendiente, Enviado, Entregado).
    - **Métodos:**
        - `cambiarEstado(String nuevoEstado)`: Cambia el estado del pedido y notifica al cliente.

Diagrama de Clases



🎯 Tarea a realizar

- Implementar la interfaz `Notificable` y las clases `Cliente` y `Pedido`.
- Cambiar el estado de un pedido y verificar que el cliente reciba la notificación.

Resumen de las Katas

1. **Kata 1**: Uso básico de interfaces para representar la capacidad de ser pagado.
2. **Kata 2**: Introducción a la herencia entre interfaces para manejar diferentes tipos de pagos con descuentos.
3. **Kata 3**: Resolución del problema del diamante utilizando interfaces para notificar al cliente sobre cambios en el estado del pedido.

Parte 2 Ejercicios sobre Excepciones

Aquí tienes la versión mejorada de los ejercicios con un formato más claro y explicaciones detalladas. He eliminado los códigos de ejemplo como solicitaste.

Ejercicios sobre Manejo de Excepciones en Java

A continuación, encontrarás cinco ejercicios diseñados para mejorar tu comprensión sobre el manejo de excepciones en Java. Practicar estos conceptos te ayudará a escribir código más robusto y seguro.

Ejercicio 2.1: División Segura

Descripción:
Crea un programa que solicite al usuario ingresar dos números enteros y realice la división del primero por el segundo. El programa debe manejar la excepción `ArithmeticException`, que ocurre si el segundo número es cero.

Objetivos:
✅ Utilizar `try-catch` para capturar y manejar la excepción.
✅ Mostrar un mensaje amigable si se detecta un intento de división por cero.
✅ Asegurar que el programa no se interrumpe abruptamente en caso de error.

Ejercicio 2.2: Conversión de Cadena a Número

Descripción:
El programa debe solicitar al usuario una cadena de texto e intentar convertirla en un número entero. Si la conversión falla porque el usuario ingresó un valor no numérico, se debe manejar la excepción `NumberFormatException`.

Objetivos:
✅ Usar `try-catch` para capturar y gestionar la excepción.
✅ Informar al usuario sobre el error de manera clara.
✅ Evitar que el programa finalice inesperadamente si la entrada no es válida.

Ejercicio 2.3: Lectura de Archivo

Descripción:
Escribe un programa que intente leer un archivo de texto y mostrar su contenido en la consola. Si el archivo no existe, debe capturar y manejar la excepción `FileNotFoundException`.

Objetivos:
✅ Utilizar `try-catch` para manejar la ausencia del archivo.
✅ Mostrar un mensaje de error si el archivo no se encuentra.
✅ Implementar una estructura de control que garantice la ejecución segura del programa.

Ejercicio 2.4: Excepción Personalizada

Descripción:
Crea una nueva excepción llamada `EdadInvalidaException` que se lance cuando un usuario ingrese una edad menor a 0 o mayor a 120. Luego, escribe un programa que solicite la edad al usuario y maneje esta excepción de manera adecuada.

Objetivos:
✅ Definir una clase de excepción personalizada que herede de `Exception`.
✅ Utilizar `throw` para lanzar la excepción si la edad ingresada es inválida.
✅ Implementar un bloque `try-catch` para manejar la excepción y mostrar un mensaje explicativo.

Ejercicio 2.5: Uso de Try-with-Resources

Descripción:
Desarrolla un programa que lea un archivo de texto utilizando `try-with-resources`, lo que garantiza el cierre automático del recurso (`BufferedReader`). El programa debe manejar la excepción `IOException` en caso de error al leer el archivo.

Objetivos:
✅ Aplicar `try-with-resources` para gestionar la apertura y cierre del archivo automáticamente.
✅ Manejar adecuadamente posibles excepciones de entrada y salida (`IOException`).
✅ Mostrar el contenido del archivo de manera ordenada en la consola.
