TECNICATURA UNIVERSITARIA
EN PROGRAMACIÓN
A DISTANCIA

PROGRAMACIÓN II

Alumno: Federico Iacono **Repo:** Link

TP 8: Interfaces y Excepciones en Java

OBJETIVO GENERAL

Desarrollar habilidades en el uso de interfaces y manejo de excepciones en Java para fomentar la modularidad, flexibilidad y robustez del código. Comprender la definición e implementación de interfaces como contratos de comportamiento y su aplicación en el diseño orientado a objetos. Aplicar jerarquías de excepciones para controlar y comunicar errores de forma segura. Diferenciar entre excepciones comprobadas y no comprobadas, y utilizar bloques `try`, `catch`, `finally` y `throw` para garantizar la integridad del programa. Integrar interfaces y manejo de excepciones en el desarrollo de aplicaciones escalables y mantenibles.

Concepto	Aplicación en el proyecto
Interfaces	Definición de contratos de comportamiento común entre distintas clases
Herencia múltiple con interfaces	Permite que una clase implementa múltiples comportamientos sin herencia de estado
Implementación de interfaces	Uso de **implements** para que una clase cumpla con los métodos definidos en una interfaz
Excepciones	Manejo de errores en tiempo de ejecución mediante estructuras **try-catch**
Excepciones checked y unchecked	Diferencias y usos según la naturaleza del error
Excepciones personalizadas	Creación de nuevas clases que extienden **Exception**
finally y try-with-resources	Buenas prácticas para liberar recursos correctamente
Uso de throw y throws	Declaración y lanzamiento de excepciones
Interfaces	Definición de contratos de comportamiento común entre distintas clases
Herencia múltiple con interfaces	Permite que una clase implementa múltiples comportamientos sin herencia de estado

Parte 1: Interfaces en un sistema de E-commerce

1. Crear una interfaz **Pagable** con el método **calcularTotal().**
2. Clase **Producto**: tiene nombre y precio, implementa **Pagable.**
3. Clase **Pedido**: tiene una lista de productos, implementa **Pagable** y calcula el total del pedido.
4. Ampliar con interfaces **Pago** y **PagoConDescuento** para distintos medios de pago (**TarjetaCredito, PayPal**), con métodos **procesarPago(double)** y **aplicarDescuento(double)**.
5. Crear una interfaz **Notificable** para notificar cambios de estado. La clase **Cliente** implementa dicha interfaz y **Pedido** debe notificarlo al cambiar de estado.

Parte 2: Ejercicios sobre Excepciones

1. **División segura**
   - Solicitar dos números y dividirlos. Manejar **ArithmeticException** si el divisor es cero.
2. **Conversión de cadena a número**
   - Leer texto del usuario e intentar convertirlo a int. Manejar **NumberFormatException** si no es válido.
3. **Lectura de archivo**
   - Leer un archivo de texto y mostrarlo. Manejar **FileNotFoundException** si el archivo no existe.
4. **Excepción personalizada**
   - Crear **EdadInvalidaException**. Lanzarla si la edad es menor a 0 o mayor a 120. Capturarla y mostrar mensaje.
5. **Uso de try-with-resources**
   - Leer un archivo con **BufferedReader** usando **try-with-resources**.

Manejar **IOException** correctamente.

CONCLUSIONES ESPERADAS

- Comprender la utilidad de las interfaces para lograr diseños desacoplados y reutilizables.
- Aplicar herencia múltiple a través de interfaces para combinar comportamientos.
- Utilizar correctamente estructuras de control de excepciones para evitar caídas del programa.
- Crear excepciones personalizadas para validar reglas de negocio.
- Aplicar buenas prácticas como **try-with-resources** y uso del bloque **finally** para manejar recursos y errores.
- Reforzar el diseño robusto y mantenible mediante la integración de interfaces y manejo de errores en Java.

Parte 1: Interfaces en un sistema de E-commerce

Inicialmente se construye el núcleo de un sistema de E-commerce utilizando interfaces para definir "contratos" de comportamiento. El objetivo es lograr un diseño modular y desacoplado.

Se crea la interfaz Pagable , que define un método calcularTotal(). Esta interfaz es implementada tanto por la clase Producto como por la clase Pedido. Aunque cada una calcula el total de forma distinta (un producto devuelve su precio y un pedido suma sus productos), el sistema puede tratar a ambos objetos simplemente como algo "Pagable".

Además, se modelan los métodos de pago (como TarjetaCredito y PayPal) usando las interfaces Pago y PagoConDescuento. Finalmente, se utiliza la interfaz Notificable para que la clase Pedido pueda informar a un Cliente sobre cambios de estado, sin necesidad de saber los detalles específicos de cómo el cliente maneja esa notificación.

Planificación

1. **Interfaz Pagable**: Se define la interfaz con el método calcularTotal().
2. **Clase Producto**: Se crea la clase con nombre y precio. Implementa Pagable, donde calcularTotal() devuelve el precio.
3. **Clase Pedido**: Se crea la clase con una lista de Producto. Implementa Pagable, donde calcularTotal() suma los totales de su lista de productos.
4. **Interfaces Pago y PagoConDescuento**: Se definen estas interfaces (procesarPago y aplicarDescuento). Se crean las clases TarjetaCredito (implementa Pago) y PayPal (implementa ambas).
5. **Interfaz Notificable**: Se define la interfaz (notificar). Se crea la clase Cliente (implementa Notificable). Se modifica Pedido para que tenga un Cliente (como Notificable) y llame a notificar() cuando cambie su estado.

Parte 2: Ejercicios sobre Excepciones

La segunda parte se enfoca en la robustez del código mediante el manejo de excepciones, asegurando que el programa pueda reaccionar a errores en tiempo de ejecución sin "romperse".

El funcionamiento se basa en el bloque **try-catch**. El código que podría fallar (como una división o la lectura de un archivo) se coloca dentro del try. Si ocurre un error, Java "lanza" una excepción que es capturada por el bloque catch correspondiente, permitiendo al programa mostrar un mensaje amigable y continuar.

Se practican varios escenarios comunes:

1. Se captura la ArithmeticException para evitar un error al dividir por cero.
2. Se maneja la NumberFormatException, que ocurre si se intenta convertir un texto no numérico (ej: "hola") a un entero.
3. Se controla la FileNotFoundException para el caso en que un archivo que se intenta leer no exista.
4. Se crea una excepción personalizada (EdadInvalidaException) para implementar reglas de negocio (validar una edad) y se lanza manualmente usando throw.

5. Se utiliza try-with-resources como una forma moderna y segura de leer archivos , asegurando que los recursos se cierren automáticamente y manejando la IOException.

Planificación

1. **División segura**: Se usa un Scanner para pedir dos números. La operación de división se envuelve en un bloque try-catch que captura ArithmeticException (si el divisor es 0) e InputMismatchException (si no se ingresan números).
2. **Conversión de cadena**: Se pide texto. Se usa Integer.parseInt() dentro de un try-catch para capturar NumberFormatException si el texto no es un entero.
3. **Lectura de archivo**: Se intenta leer un archivo (p.ej., archivo.txt) usando FileReader y Scanner (o BufferedReader). Se envuelve en try-catch para FileNotFoundException.
4. **Excepción personalizada**: Se crea la clase EdadInvalidaException que hereda de Exception. Se crea un método validarEdad(int edad) que lanza (throw) esta excepción si la edad está fuera del rango (0-120). En el main (o método de prueba), se llama a este método dentro de un try-catch.
5. **Try-with-resources**: Se lee un archivo (p.ej., datos.txt) usando BufferedReader y FileReader declarados dentro de los paréntesis del try (...). Esto asegura que los recursos se cierren solos. Se captura IOException.

Repositorio con la implementación en JAVA: Link