

DEBER 9

Nombre: Vasquez Mera Juan Daniel

1. Realizar un proyecto por cada ejercicio y adjuntar el enlace del repositorio Github(1 repositorio por cada ejercicio).
2. Al ejercicio 2 realizalo a mano, adjuntar capturas claras y con buena letra escaneadas, agrégalas al final de este documento.
3. Escoger un ejercicio de cualquiera de los 3 planteados y grabarse programando, debe escucharse la voz claramente del estudiante mientras explica lo que realiza y su lógica, debe verse claramente el código y con buena definición. El video no debe durar más de 15 minutos y ser subido de forma pública a Youtube. Sino se cumple cualquiera de estos criterios **no se dará por valido el item 3 que pondra la mitad del deber.**

Ejercicio 1: Sistema de Cuentas Bancarias

Contexto

Un banco necesita manejar diferentes tipos de cuentas: **Cuenta de Ahorros** y **Cuenta Corriente**. Ambas comparten operaciones básicas, pero tienen comportamientos específicos.

Requisitos

Interfaces a crear:

1. Interface OperacionesBancarias

- void depositar(double monto)
- boolean retirar(double monto)
- double consultarSaldo()

2. Interface Transferible

- boolean transferir(double monto, String cuentaDestino)
-

Clases a implementar:

1. Clase CuentaAhorros (implementa OperacionesBancarias)

- Atributos: numeroCuenta, titular, saldo
- **Restricción:** No puede tener saldo negativo

- **Regla:** Si el saldo cae por debajo de \$100, cobra comisión de \$5

2. Clase CuentaCorriente (implementa OperacionesBancarias y Transferible)

- Atributos: numeroCuenta, titular, saldo, sobregiro (límite: \$500)
 - **Ventaja:** Puede tener saldo negativo hasta -\$500 (sobregiro)
 - **Funcionalidad:** Permite transferencias a otras cuentas
-

Tareas

1. Crear las 2 interfaces
 2. Implementar las 2 clases con sus reglas específicas
 3. Crear un main que:
 - Cree 1 cuenta de ahorros con \$200
 - Cree 1 cuenta corriente con \$1000
 - Realice las siguientes operaciones:
 - Depositar \$300 en cuenta de ahorros
 - Retirar \$450 de cuenta de ahorros (mostrar comisión)
 - Retirar \$1200 de cuenta corriente (usar sobregiro)
 - Transferir \$300 desde cuenta corriente a otra cuenta
 - Mostrar saldos finales
-

Salida Esperada

==== CUENTA DE AHORROS ===

Depósito: \$300.0 - Saldo: \$500.0

Retiro: \$450.0 - Saldo: \$50.0

Comisión aplicada: \$5.0 - Saldo: \$45.0

==== CUENTA CORRIENTE ===

Retiro: \$1200.0 - Saldo: \$-200.0 (Sobregiro usado)

Transferencia: \$300.0 a cuenta 9876 - Saldo: \$-500.0

==== SALDOS FINALES ===

Cuenta Ahorros: \$45.0

Cuenta Corriente: \$-500.0

Pista de implementación

```
public class CuentaAhorros implements OperacionesBancarias{  
    private String numeroCuenta;  
    private String titular;  
    private double saldo;  
  
    @Override  
    public boolean retirar(double monto){  
        if (saldo >= monto) {  
            saldo -= monto;  
            // Verificar si aplica comisión  
            if (saldo < 100) {  
                saldo -= 5;  
                System.out.println("Comisión aplicada: $5.0");  
            }  
            return true;  
        }  
        return false;  
    }  
    // ... resto de métodos  
}
```

Enlace de Github del ejercicio
[Ejercicio 1- Sistema de Cuentas Bancarias](#)

Ejercicio 2: Sistema de Autenticación Multi-Factor(usar una lista)

Contexto

Una aplicación empresarial necesita implementar autenticación de usuarios con diferentes niveles de seguridad: Autenticación Básica (usuario/contraseña), Autenticación con OTP (código de un solo uso) y Autenticación Biométrica.

Requisitos

Interfaces a crear:

1. Interface Autenticable

- boolean autenticar(String usuario, String credencial)
- void cerrarSesion()
- boolean sesionActiva()

2. Interface MultiFactor

- String generarCodigoVerificacion()
- boolean verificarCodigo(String codigo)
- int intentosRestantes()

3. Interface Auditable

- void registrarIntento(String usuario, boolean exitoso)
 - List<String> obtenerHistorial(String usuario)
-

Clases a implementar:

1. Clase AutenticacionBasica (implementa Autenticable y Auditabile)

- Atributos: usuario, password, sesionActiva
- Validación simple: usuario y password coinciden
- Registra todos los intentos (exitosos y fallidos)
- Bloquea después de 3 intentos fallidos

2. Clase AutenticacionOTP (implementa Autenticable, MultiFactor y Auditabile)

- Atributos: usuario, password, codigoOTP, intentosRestantes
- Genera código de 6 dígitos aleatorio

- Requiere usuario/password + código OTP
- Permite 3 intentos para el código
- Registra todos los intentos

3. Clase AutenticacionBiometrica (implementa Autenticable y Auditabile)

- Atributos: usuario, huellaDactilar, reconocimientoFacial
 - Simula verificación biométrica (comparación de strings)
 - No requiere password tradicional
 - Registra con nivel de confianza (alto/medio/bajo)
-

Tareas

1. Crear las 3 interfaces
 2. Implementar las 3 clases de autenticación
 3. Crear un main que simule:
 - o Login básico exitoso y uno fallido
 - o Login con OTP (generar código, verificar)
 - o Login biométrico con huella
 - o Mostrar historial de intentos
 - o Intentar login después de 3 fallos (bloqueo)
-

Salida Esperada

==== SISTEMA DE AUTENTICACIÓN ===

--- Autenticación Básica ---

Usuario:admin / Password:admin123

✓ Autenticación exitosa

[AUDIT] Intento exitoso:admin - 2024-11-27 10:30:15

Usuario:admin / Password:wrongpass

X Autenticación fallida

[AUDIT] Intento fallido: admin - 2024-11-27 10:30:20

--- Autenticación OTP ---

Usuario: juan.gonzalez / Password: secure456

Código OTP generado: 784523

Código enviado por SMS

Ingresando código: 784523

✓ Código verificado correctamente

✓ Autenticación OTP exitosa

[AUDIT] Intento exitoso: juan.gonzalez (OTP) - 2024-11-27 10:31:00

--- Autenticación Biométrica ---

Usuario: maria.lopez

Escaneando huella dactilar...

✓ Huella verificada - Confianza: ALTA

✓ Autenticación biométrica exitosa

[AUDIT] Intento exitoso: maria.lopez (Biométrica/Alta) - 2024-11-27 10:31:30

--- Historial de Usuario: admin ---

[2024-11-27 10:30:15] LOGIN EXITOSO

[2024-11-27 10:30:20] LOGIN FALLIDO

--- Prueba de Bloqueo ---

Intento 1: X Fallido - Intentos restantes: 2

Intento 2: X Fallido - Intentos restantes: 1

Intento 3: X Fallido - Intentos restantes: 0

CUENTA BLOQUEADA - Contacte al administrador

Pista de implementación

```
public class AutenticacionOTP implements Autenticable, MultiFactor, Auditable {  
    private String usuario;  
    private String password;  
    private String codigoOTP;  
    private int intentos;  
    private boolean sesionActiva;  
    private List<String> historial;  
  
    public String generarCodigoVerificacion(){  
        // Genera número aleatorio de 6 dígitos  
        codigoOTP = String.format("%06d", (int)(Math.random() * 1000000));  
        intentos = 3;  
        System.out.println("Código OTP generado: " + codigoOTP);  
        System.out.println("Código enviado por SMS");  
        return codigoOTP;  
    }  
  
    @Override  
    public boolean verificarCodigo(String codigo){  
        if (intentos <= 0) {  
            System.out.println("X Sin intentos restantes");  
            return false;  
        }  
  
        intentos--;  
        if (codigoOTP.equals(codigo)) {  
            System.out.println("Código válido");  
            return true;  
        } else {  
            System.out.println("Código inválido");  
            return false;  
        }  
    }  
}
```

```

        System.out.println("✓ Código verificado correctamente");

        return true;

    } else {

        System.out.println("✗ Código incorrecto - Intentos restantes: " + intentos);

        return false;
    }
}

// ... resto de métodos

}

```

[Enlace de Github del ejercicio](#)
[Ejercicio 2 - Sistema de Autenticacion Multi-Factor](#)

Ejercicio 3: Sistema de Procesamiento de Pagos

Contexto

Una plataforma de comercio electrónico necesita integrar múltiples métodos de pago: Tarjeta de Crédito, PayPal y Transferencia Bancaria. Cada método tiene diferentes requisitos de validación y procesamiento.

Requisitos

Interfaces a crear:

1. Interface MetodoPago

- boolean validar()
- boolean procesarPago(double monto, String referencia)
- String generarComprobante()

2. Interface Reembolsable

- boolean procesarDevolucion(double monto, String motivo)
- int diasParaDevolucion() // días permitidos para devolución

3. Interface Verificable

- boolean verificarIdentidad(String documento)
 - boolean esSeguro()
-

Clases a implementar:

1. Clase PagoTarjeta (implementa MetodoPago, Reembolsable y Verifiable)

- Atributos: numeroTarjeta, cvv, fechaExpiracion, titular
- Validaciones: CVV (3 dígitos), fecha no expirada
- Devolución: hasta 30 días
- Requiere verificación de identidad

2. Clase PagoPayPal (implementa MetodoPago y Reembolsable)

- Atributos: email, token
- Validación: email válido (contiene @)
- Devolución: hasta 180 días
- No requiere verificación adicional

3. Clase PagoTransferencia (implementa MetodoPago y Verifiable)

- Atributos: numeroCuenta, banco, titular
 - Validación: número de cuenta (10 dígitos)
 - No permite devoluciones
 - Requiere verificación de identidad
-

Tareas

1. Crear las 3 interfaces
2. Implementar las 3 clases de métodos de pago
3. Crear un main que simule:
 - Compra de \$250 con tarjeta de crédito
 - Compra de \$180 con PayPal
 - Compra de \$500 con transferencia
 - Intentar devolución en tarjeta (\$50)

- Intentar devolución en transferencia (debe fallar)
 - Mostrar comprobantes generados
-

Salida Esperada

==== PROCESAMIENTO DE PAGOS ===

--- Pago con Tarjeta de Crédito ---

- ✓ Validación exitosa
 - ✓ Identidad verificada: CI 1234567890
 - ✓ Método de pago seguro
- Pago procesado: \$250.0 - Ref: TXN-001

Comprobante: TARJETA-xxxx-5678-TXN-001

--- Pago con PayPal ---

- ✓ Validación exitosa
- Pago procesado: \$180.0 - Ref: TXN-002
- Comprobante: PAYPAL-usuario@email.com-TXN-002

--- Pago con Transferencia ---

- ✓ Validación exitosa
 - ✓ Identidad verificada: CI 0987654321
 - ✓ Método de pago seguro
- Pago procesado: \$500.0 - Ref: TXN-003
- Comprobante: TRANSFER-1234567890-TXN-003

==== PROCESAMIENTO DE DEVOLUCIONES ===

--- Devolución Tarjeta ---

✓ Devolución procesada: \$50.0

Motivo: Producto defectuoso

Días permitidos: 30 días

--- Devolución Transferencia ---

X Este método de pago no permite devoluciones

TOTAL PROCESADO: \$930.0

TOTAL DEVUELTO: \$50.0

NETO: \$880.0

Enlace de Github del ejercicio

[Ejercicio 3: Sistema de Procesamiento de Pagos](#)

Ejercicio a Mano aquí adjuntar capturas o fotos.

Juan Vásquez GR2 Ejercicio 2 Sistema de Autenticación Multi-Factor

```
public interface Autenticable {
    boolean autenticar (String usuario, String credencial);
    void cerrarSesion();
    boolean sesionActiva();
}

public interface MultiFactor {
    String generarCodigoVerificacion();
    boolean verificarCodigo (String codigo);
    int intentosRestantes();
}

import java.util.List;

public interface Auditable {
    void registrarIntento (String usuario, boolean exitoso);
    List<String> obtenerHistorial (String usuario);
}

import java.util.ArrayList;
import java.util.List;

public class AutenticacionBasica implements Autenticable, Auditable {
    private String usuarioValido;
    private String passwordValido;
    private boolean sesionActiva;
    private List<String> historial;
    private int intentosFallidos;

    public AutenticacionBasica (String usuario, String password) {
        this.usuarioValido = usuario;
        this.passwordValido = password;
        this.historial = new ArrayList<>();
        this.intentosFallidos = 0;
    }

    @Override
    public boolean autenticar (String usuario, String credencial) {
        if (intentosFallidos >= 3) {
            System.out.println ("Cuenta Bloqueada - Contacte al administrador");
            return false;
        }
        if (usuarioValido.equals (usuario) && passwordValido.equals (credencial)) {
            sesionActiva = true;
            System.out.println ("Autenticación exitosa");
            registrarIntento (usuario, true);
            intentosFallidos = 0;
            return true;
        } else {
            intentosFallidos++;
            System.out.println ("Autenticación Fallida");
            if (intentosFallidos < 3) {
                System.out.println ("Intento " + intentosFallidos + " Fallido");
            }
        }
    }
}
```

```

        registrarIntento(usuario, false);
        return false;
    }

    @Override
    public void cerrarSession() {
        sessionActiva = false;
    }

    @Override
    public boolean sessionActiva() {
        return sessionActiva;
    }

    @Override
    public void registrarIntento (String usuario, boolean exitoso) {
        String fecha = "2024-12-04 10:30:15";
        String estado = exitoso ? "LOGIN EXITOSO" : "LOGIN FALLIDO";
        historial.add ("[" + fecha + "] " + estado);
        String tipo = exitoso ? "Intento Exitoso" : "Intento Fallido";
        System.out.println ("[AUDIT] " + tipo + ": " + usuario + " - " + fecha);
    }

    @Override
    public List<String> obtenerHistorial (String usuario) {
        return historial;
    }

    import java.util.ArrayList;
    import java.util.List;

    public class AutenticacionOTP implements Autenticable, MultiFactor, Auditatable {
        private String usuarioValido;
        private String passwordValido;
        private String codigoOTP;
        private int intentosCuidados;
        private boolean sessionActiva;
        private List<String> historial;

        public AutenticacionOTP (String usuario, String password) {
            this.usuarioValido = usuario;
            this.passwordValido = password;
            this.historial = new ArrayList<>();
        }

        @Override
        public boolean autenticar (String usuario, String credencial) {
            return usuarioValido.equals(usuario) && passwordValido.equals(credencial);
        }
    }

```

@Desarrollador

```
public String generarCodigoVerificacion() {
    codigoOTP = "784523";
    intentosCodigo = 3;
    System.out.println("Codigo OTP generado: " + codigoOTP);
    System.out.println("Codigo enviado por SMS");
    return codigoOTP;
}
```

@Desarrollador

```
public boolean verificarCodigo(String codigo) {
    if (intentosCodigo <= 0) {
        System.out.println("Ingresando código - Sin intentos restantes");
        return false;
    }
    intentos_codigo--;
    System.out.println("Ingresando código: " + codigo);
    if (codigoOTP.equals(codigo)) {
        System.out.println("Código verificado correctamente");
        sessionActiva = true;
        System.out.println("Autenticación OTP exitosa");
        registrarIntento(usuarioValido, true);
        return true;
    } else {
        System.out.println("Código incorrecto");
        registrarIntento(usuarioValido, false);
        return false;
    }
}
```

@Desarrollador

```
public int intentosRestantes() {
    return intentosCodigo;
}
```

@Desarrollador

```
public void cerrarSession() {
    sessionActiva = false;
}
```

@Desarrollador

```
public boolean sessionActiva() {
    return sessionActiva;
}
```

@Desarrollador

```
public void registrarIntento(String usuario, boolean exitoso) {
    String fecha = "2024-12-01 10:31:00";
    System.out.println("[AUDIT] Intento exitoso: " + usuario + " (" + (exitoso ? "OK" : "FAIL") + ")");
    historial.add("[" + fecha + "] [login OTP] " + (exitoso ? "exitoso" : "fallido"));
}
```

@Override

```
public List<String> obtenerHistorial(String usuario) {  
    return historial;  
}
```

```
import java.util.ArrayList;  
import java.util.List;
```

```
public class AutenticacionBiometrica implements Autenticable, Auditable {
```

```
private String usuarioValido;  
private String huellaRegistrada;  
private boolean sessionActiva;  
private List<String> historial;
```

```
public AutenticacionBiometrica (String usuario, String huella) {
```

```
    this.usuarioValido = usuario;  
    this.huellaRegistrada = huella;  
    this.historial = new ArrayList<>();  
}
```

@Override

```
public boolean autenticar (String usuario, String huella) {
```

```
    System.out.println ("Examinando huella dactilar...");
```

```
    if (usuarioValido.equals(usuario) && huellaRegistrada.equals(huella)) {
```

```
        sessionActiva = true;
```

```
        System.out.println ("Huella verificada - Confianza: ALTA");
```

```
        System.out.println ("Autenticacion biometrica exitosa");
```

```
        registrarIntento (usuario, true);
```

```
        return true;
```

```
}
```

```
    registrarIntento (usuario, false);
```

```
    return false;
```

```
}
```

@Override

```
public void cerrarSession () {
```

```
    sessionActiva = false;
```

```
}
```

@Override

```
public boolean sessionActiva () {
```

```
    return sessionActiva;
```

```
}
```

@Override

```
public void registrarIntento (String usuario, boolean exitoso) {
```

```
    String fecha = "2024-12-04 10:30:31";  
    System.out.println ("[UDIY] Intento exitoso: " + usuario + " Biometria (alta) - " + fecha);  
    historial.add ("[" + fecha + "] [UDIY Biometrico] " + exitoso ? "Exito": "FALSO");
```

```
}
```

@Override

```
public List<String> obtenerHistorial (String usuario) { return historial; }
```

```
}
```

```

import java.util.List;
public class Main {
    public static void main (String [] args) {
        System.out.println ("==> SISTEMA DE AUTENTIFICACION ==>");
        System.out.println ("--- Autenticacion Basica ---");
        AutenticacionBasica basicAuth = new AutenticacionBasica ("sysadmin", "SuperClave99");
        System.out.println ("Usuario: sysadmin / Password: SuperClave99");
        basicAuth.autenticar ("sysadmin", "SuperClave99");
        System.out.println ("Usuario: sysadmin / Password: ClaveIncorrecta");
        basicAuth.autenticar ("sysadmin", "ClaveIncorrecta");
        System.out.println ("--- Autenticacion OTP ---");
        AutenticacionOTP otpAuth = new AutenticacionOTP ("roberto.p", "pass1234");
        System.out.println ("Usuario: roberto.p (Password: pass1234)");
        if (otpAuth.autenticar ("roberto.p", "pass1234")) {
            String codigo = otpAuth.generarCodigoVerificacion ();
            otpAuth.verificarCodigo (codigo);
        }
        System.out.println ("--- Autenticacion Biometrica ---");
        AutenticacionBiometrica bioAuth = new AutenticacionBiometrica ("elena.s", "face.id.data");
        System.out.println ("Usuario: elena.s");
        bioAuth.autenticar ("elena.s", "face.id.data");
        System.out.println ("--- Historial de usuario: sysadmin");
        List<String> historial = basicAuth.obtenerHistorial ("sysadmin");
        for (String log : historial) {
            System.out.println (log);
        }
        System.out.println ("--- PRUEBA DE BLOQUEO ---");
        basicAuth.autenticar ("sysadmin", "error1");
        basicAuth.autenticar ("sysadmin", "error2");
        basicAuth.autenticar ("sysadmin", "error3");
    }
}

```

Adjunta enlace del video a youtube, debe estar público.

<https://youtu.be/akVYwTShONs>