Nombre Alumno/s: Scetti, Santiago **DNI:** 43752065
Varela, Franco Martín 44280211

Nombre Profesor:

Grupo Laboratorio: "3" **TP:** 5 **Fecha de entrega:** 20/10/2024

Introducción:

A lo largo se de este informe, se detallarán los conceptos teóricos desarrollados en las clases de teoría junto a la puesta en práctica del trabajo práctico correspondiente. El ensayo busca responder a conceptos avanzados del paradigma orientado a objetos, como *herencia* y *polimorfismo*, favoreciendo a la robustez del código y a la reutilización del mismo.

Herencia: Refiere al mecanismo de compartir atributos y métodos. Las superclases representan abstracciones generalizadas, y las subclases representan especializaciones.

Herencia Simple: Es el mecanismo por el cual una clase puede extender las características de otra, pero puede heredar o tomar atributos de una sola clase padre. En ocasiones es necesario redefinir métodos de la super clase, de modo que métodos con el mismo nombre y características se comporten de forma distinta en la subclase, basta con declarar un método miembro con la misma firma.

```java
public class Elipse extends FiguraGeometrica{

    private double sEjeMayor;
    private double sEjeMenor;

    public Elipse(double p_sEjeMayor, double p_sEjeMenor, Punto p_centro){
        super(p_centro);
        setSEjeMayor(p_sEjeMayor);
        setSEjeMenor(p_sEjeMenor);
    }
     public Elipse(double p_sEjeMayor, double p_sEjeMenor){
        super(new Punto(0,0));
        setSEjeMayor(p_sEjeMayor);
        setSEjeMenor(p_sEjeMenor);
    }
```

Polimorfismo: En lenguajes de programación estructurados en clases, se refiere a la posibilidad de que objetos de clases diferentes respondan a mensajes con el mismo nombre, cada uno de ellos con su propio comportamiento. El *Override* es una técnica de la POO que permite darle especificidad al comportamiento de una clase, cuando éste no es exactamente igual al de la clase ancestro, pero manteniendo la semántica.

```java
public class CajaDeAhorro extends CuentaBancaria{

    // Variables de instancia
    private int extraccionesPosibles;


    // Constructores
    public CajaDeAhorro(int p_nroCuenta, Persona p_titular, double p_saldo) {
        super(p_nroCuenta, p_titular, p_saldo);
        setExtraccionesPosibles(10);
    }

    public CajaDeAhorro(int p_nroCuenta, Persona p_titular) {
        super(p_nroCuenta, p_titular, 0);
        setExtraccionesPosibles(10);
    }

    // SETTERS
    private void setExtraccionesPosibles(int p_extracciones) {
        this.extraccionesPosibles = p_extracciones;
    }
```

```java
public class CuentaCorriente extends CuentaBancaria{

    // instance variables
    private double limiteDescubierto;

    // Contructores
    public CuentaCorriente(int p_nroCuenta, Persona p_titular, double p_saldo) {
        super(p_nroCuenta, p_titular, p_saldo);
        setLimiteDescubierto(500.0);
    }
```

Nombre Alumno/s: Scetti, Santiago **DNI:** 43752065
Varela, Franco Martín 44280211

Nombre Profesor:

Grupo Laboratorio: "3" **TP:** 5 **Fecha de entrega:** 20/10/2024

En este ejemplo, CuentaBancaria al ser abstracta puede comportarse tanto como una Cuenta Corriente o una Caja de Ahorro. Así mismo, CuentaBancaria puede ser utilizada por la aplicación Banco sin necesidad de declarar o cuestionar de qué tipo se trata.

```
public abstract class CuentaBancaria {
    private int nroCuenta;
    private double saldo;
    private Persona titular; // (1)

    public CuentaBancaria(int p_nroCuenta, Persona p_titular, double p_saldo) {
        this.setNroCuenta(p_nroCuenta);
        this.setTitular(p_titular);
        this.setSaldo(p_saldo);    Overridable method call in constructor
    }
```

Ejemplo de la utilización de Sobreescritura

```
@Override
public String tipoVisitante(){
    return "Visitante";
}

@Override
public void mostrar(){
    System.out.println("Nombre y Apellido: " + persona.getNombre() + "\n" +
    "DNI: " + persona.getDNI() + "Edad: " + persona.edad() + " Años");
}

@Override
public double entrada(){
    return 10;
}
```