	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorio de Computacion Salas A y B

Profesor(a): Jose Antonio Ayala Barbosa

Asignatura: Programación orientada a objetos

Grupo: 02

No de practica(s): 8

Integrante(s): Santos Carmona Valeria Sofia | Hernandez Reyes Rebeca Sarai |Al

No de lista o brigada:

Semestre: 2026-1

Fecha de entrega: Jueves 23 de Octubre 2025

Observaciones:

Calificacion:

PRÁCTICA 8 Polimorfismo

Integrantes : Santos Carmona Valeria Sofia | Hernandez Reyes Rebeca Sarai | Almaguer Miranda Michel Santiago

I. OBJETIVO

Implementar el concepto de polimorfismo en un lenguaje de programación orientado a objetos.

ACTIVIDADES

A partir de una jerarquía de clases, implementar referencias que se comporten como diferentes objetos.

CÓDIGO FUENTE Y DESARROLLO

El proyecto se desarrolló en Java utilizando Apache NetBeans.

Clase polimorfismo :

El propósito de esta clase es demostrar el polimorfismo utilizando clases e interfaces definidas. El Main crea instancias de triangulo y el cuadrilátero y les asigna variables de tipo polígono.

Se manda a llamar a los métodos area y perímetro en las variables del tipo polígono , el polimorfismo es el que hace el trabajo de asegurarse de que se llame a la implementación correcta para cada uno de los objetos.

Lo mismo pasa para flauta y tipoIntrumento en la variable IntrumnetoMusical donde el polimorfismo se aplica asegurándose de que se llame la implementación correcta para la flauta.

```
package polimorfismo;

public class Polimorfismo {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        // Polimorfismo en acción
        Poligono poligonoTriangulo = new Triangulo(60, 60, 60, 5, 5, 5, 4, 3);
        Poligono poligonoCuadrilatero = new Cuadrilatero(90, 90, 4, 4, 4, 4);

        System.out.println(poligonoTriangulo);
        System.out.println("Área: " + poligonoTriangulo.area());
        System.out.println("Perímetro: " + poligonoTriangulo.perimetro());

        System.out.println(poligonoCuadrilatero);
        System.out.println("Área: " + poligonoCuadrilatero.area());
        System.out.println("Perímetro: " + poligonoCuadrilatero.perimetro());

        /**
         * instrumentos
         */
        InstrumentoMusical instrumento = new Flauta();
        instrumento.tocar();
        System.out.println(instrumento.tipoInstrumento());
    }

}
```

- public class Polimorfismo : Define la clase principal llamada Polimorfismo. La palabra clave public significa que esta clase puede ser accedida desde cualquier otra clase.
- Poligono poligonoTriangulo = new Triangulo(60, 60, 60, 5, 5, 5, 4, 3);: Crea un objeto de la clase Triangulo y lo asigna a una variable de tipo Poligono. Aquí es donde entra en juego el polimorfismo: Una variable de tipo Poligono (la clase padre) puede referenciar un objeto de tipo Triangulo (una de las clases hijas).
- Poligono poligonoCuadrilatero = new Cuadrilatero(90, 90, 4, 4, 4, 4);: Similar al anterior, crea un objeto Cuadrilatero y lo asigna a una variable de tipo Poligono.
- System.out.println(poligonoTriangulo);: Imprime la representación de cadena del objeto poligonoTriangulo. Como Triangulo hereda de Poligono (y no se ha sobreescrito el método toString), se imprimirá "Triángulo".
- System.out.println("Área: " + poligonoTriangulo.area());: Llama al método area() del objeto poligonoTriangulo. El polimorfismo asegura que se llame a la implementación correcta del método area() para un Triangulo.
- System.out.println("Perímetro: " + poligonoTriangulo.perimetro());: Llama al método perimetro() del objeto poligonoTriangulo. De nuevo, el polimorfismo asegura que se llame a la implementación correcta.

- System.out.println(...) (para poligonoCuadrilatero): Realiza las mismas operaciones que para el triángulo, pero con un objeto Cuadrilatero. El polimorfismo permite que el mismo código funcione correctamente con diferentes tipos de objetos.
- InstrumentoMusical instrumento = new Flauta();: Crea un objeto de la clase Flauta y lo asigna a una variable de tipo InstrumentoMusical. Aquí también se aplica el polimorfismo: Una variable de tipo InstrumentoMusical (la interfaz) puede referenciar un objeto de tipo Flauta (una clase que implementa la interfaz).
- instrumento.tocar();: Llama al método tocar() del objeto instrumento. El polimorfismo permite que se llame a la implementación correcta del método tocar() para una Flauta.
- System.out.println(instrumento.tipoInstrumento());: Llama al método tipoInstrumento() del objeto instrumento. El polimorfismo asegura que se llame a la implementación correcta.

Clase Poligono

Su propósito es definir la base de todas las figuras geométricas , al ser abstracta no se puede crear directamente un objeto desde ella , contiene dentro los métodos abstracto area y perímetro que eran implementados por las subclases.

```java
package polimorfismo;


public abstract class Poligono {
    public abstract double area();
    public abstract double perimetro();

    @Override
    public String toString() {
        return "Polígono";
    }
}
```

- public abstract class Poligono { ... }: Define una clase abstracta llamada Poligono. Una clase abstracta no se puede instanciar directamente (no puedes crear un objeto Poligonodirectamente).
- public abstract double area();: Define un método abstracto llamado area(). Un método abstracto no tiene implementación en la clase abstracta. Las subclases concretas (no abstractas) de Poligono deben implementar este método.
- public abstract double perimetro();: Define un método abstracto llamado perimetro(). Similar al método area(), las subclases concretas deben implementar este método.
- @Override public String toString() { ... }: Sobreescribe el método toString() de la clase Object. Este método devuelve una representación de cadena del objeto. En este caso, simplemente devuelve “Polígono".

Clase triangulo

Su proposito es representar al objeto triangulo y calcular su area y perimetro. Hereda de polígono e implementa los métodos area y perímetro para el triangulo , Contiene variables de instancia los cuales almacenan información sobre los ángulos , lados , base y altura.

También contiene un Constructor que inicializa las variables anteriores.

```
package polimorfismo;

public class Triangulo extends Poligono {
    private int alfa, beta, gama;
    private float a, b, c;
    private float base, altura;

    public Triangulo(int alfa, int beta, int gama, float a, float b, float c, float base, float altura) {
        this.alfa = alfa;
        this.beta = beta;
        this.gama = gama;
        this.a = a;
        this.b = b;
        this.c = c;
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double area() {
        return (base * altura) / 2;
    }

    @Override
    public double perimetro() {
        return a + b + c;
    }

    @Override
    public String toString() {
        return "Triángulo";
    }
}
```

- public class Triangulo extends Poligono { ... }: Define una clase llamada Triangulo que hereda de la clase Poligono. Esto significa que Triangulo hereda todas las propiedades y métodos de Poligono.
- private int alfa, beta, gama;: Define variables de instancia privadas para almacenar los ángulos del triángulo.
- private float a, b, c;: Define variables de instancia privadas para almacenar los lados del triángulo.
- private float base, altura;: Define variables de instancia privadas para almacenar la base y la altura del triángulo.
- public Triangulo(int alfa, int beta, int gama, float a, float b, float c, float base, float altura) { ... }: Define el constructor de la clase Triangulo. El constructor se utiliza para inicializar los valores de las variables de instancia cuando se crea un nuevo objeto Triangulo.
    - this.alfa = alfa;: Asigna el valor del parámetro alfa a la variable de instancia alfa. La palabra clave this se utiliza para referirse a la variable de instancia de la clase.
    - Las asignaciones restantes (this.beta = beta;, this.gama = gama;, etc.) hacen lo mismo para las demás variables de instancia.
- @Override public double area() { ... }: Implementa el método abstracto area() de la clase Poligono. Proporciona la implementación específica para calcular el área de un triángulo: (base * altura) / 2.
    - @Override: Indica que este método está sobreescribiendo un método de la clase padre (Poligono).
    - public double area(): Define el método area() que devuelve un valor de tipo double (un número de punto flotante de doble precisión).
    - return (base * altura) / 2;: Calcula el área del triángulo y devuelve el resultado.
- @Override public double perimetro() { ... }: Implementa el método abstracto perimetro() de la clase Poligono. Proporciona la implementación específica para calcular el perímetro de un triángulo: a + b + c.
- @Override public String toString() { ... }: Sobreescribe el método toString() de la clase Object. Este método devuelve una representación de cadena del objeto. En este caso, devuelve "Triángulo".

Clase Cuadrilátero

Su propósito es representar al objeto cuadrilátero y calcula area y perímetro , implementa los métodos area y perímetro específicos para este objeto que hereda de polígono , contiene variables de instancia donde se almacena la información sobre ángulos , base y altura y tiene el constructor que inicializa las variables de instancia.

```java
package polimorfismo;

public class Cuadrilatero extends Poligono {
    private int alfa, beta;
    private float a, b;
    private float base, altura;

    public Cuadrilatero(int alfa, int beta, float a, float b, float base, float altura) {
        this.alfa = alfa;
        this.beta = beta;
        this.a = a;
        this.b = b;
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double area() {
        return base * altura;
    }

    @Override
    public double perimetro() {
        return 2 * (a + b);
    }

    @Override
    public String toString() {
        return "Cuadrilátero";
    }
}
```

- public class Cuadrilatero extends Poligono { ... }: Define una clase llamada Cuadrilateroque hereda de la clase Poligono. Esto significa que Cuadrilatero hereda todas las propiedades y métodos de Poligono.
- private int alfa, beta;: Define variables de instancia privadas para almacenar dos ángulos del cuadrilátero (podrían representar, por ejemplo, dos ángulos opuestos).
- private float a, b;: Define variables de instancia privadas para almacenar dos lados del cuadrilátero .
- private float base, altura;: Define variables de instancia privadas para almacenar la base y la altura del cuadrilátero.
- public Cuadrilatero(int alfa, int beta, float a, float b, float base, float altura) { ... }: Define el constructor de la clase Cuadrilatero. El constructor se utiliza para inicializar los valores de las variables de instancia cuando se crea un nuevo objeto Cuadrilatero.
    - this.alfa = alfa;: Asigna el valor del parámetro alfa a la variable de instancia alfa. La palabra clave this se utiliza para referirse a la variable de instancia de la clase.
    - Las asignaciones restantes (this.beta = beta;, this.a = a;, etc.) hacen lo mismo para las demás variables de instancia.
- @Override public double area() { ... }: Implementa el método abstracto area() de la clase Poligono. Proporciona la implementación específica para calcular el área de un cuadrilátero (en este caso, se asume que es un rectángulo o un paralelogramo): base * altura.
    - @Override: Indica que este método está sobreescribiendo un método de la clase padre (Poligono).
    - public double area(): Define el método area() que devuelve un valor de tipo double (un número de punto flotante de doble precisión).
    - return base * altura;: Calcula el área del cuadrilátero y devuelve el resultado.
- @Override public double perimetro() { ... }: Implementa el método abstracto perimetro() de la clase Poligono. Proporciona la implementación específica para calcular el perímetro de un cuadrilátero (en este caso, se asume que es un rectángulo o un paralelogramo): 2 * (a + b).
- @Override public String toString() { ... }: Sobreescribe el método toString() de la clase Object. Este método devuelve una representación de cadena del objeto. En este caso, devuelve "Cuadrilátero".

Clase InstrumentoMusical

Su propósito es definir el contrato para cualquier clase que representa a un instrumento musical en este caso la flauta ademas de declarar los métodos de tocar y afinar.

```
package polimorfismo;

/**
 *
 * @author christiansantosflores
 */
interface InstrumentoMusical {
    void tocar();
    void afinar();
    String tipoInstrumento();
}
```

- interface InstrumentoMusical { ... }: Define una interfaz llamada InstrumentoMusical. Una interfaz define un conjunto de métodos que una clase debe implementar si dice que implementa esa interfaz.
- void tocar();: Define un método llamado tocar que no toma argumentos y no devuelve ningún valor (void). Cualquier clase que implemente InstrumentoMusical debe proporcionar una implementación para este método.
- void afinar();: Define un método llamado afinar que no toma argumentos y no devuelve ningún valor.
- String tipoInstrumento();: Define un método llamado tipoInstrumento que no toma argumentos y devuelve una cadena (String).

Clase InstrumentoViento

Su propósito es dar una implementación base para los instrumentos de viento ademas de implementar InstrumentoMusical donde se implementan los métodos tocar , afinar y tipoInstrumento con comportamientos básicos para este .

```
package polimorfismo;

/**
 *
 * @author christiansantosflores
 */
class InstrumentoViento implements InstrumentoMusical {
    @Override
    public void tocar() {
        System.out.println("Tocando instrumento de viento.");
    }

    @Override
    public void afinar() {
        System.out.println("Afinando instrumento de viento.");
    }

    @Override
    public String tipoInstrumento() {
        return "Instrumento de viento";
    }
}
```

- class InstrumentoViento implements InstrumentoMusical { ... }: Define una clase llamada InstrumentoViento que implementa la interfaz InstrumentoMusical. Esto significa que InstrumentoViento debe proporcionar una implementación para todos los métodos definidos en la interfaz InstrumentoMusical.
- @Override public void tocar() { ... }: Implementa el método tocar de la interfaz InstrumentoMusical. En este caso, simplemente imprime un mensaje en la consola.
- @Override public void afinar() { ... }: Implementa el método afinar de la interfaz InstrumentoMusical.
- @Override public String tipoInstrumento() { ... }: Implementa el método tipoInstrumento de la interfaz InstrumentoMusical. Devuelve la cadena "Instrumento de viento”.

Clse Flauta

Representa el objeto flauta específicamente y hereda la implementación de tocar y afinar de InstrumentoViento . Sobreescribe tipoInstrumento que devuelve flauta

```java
package polimorfismo;

/**
 *
 * @author christiansantosflores
 */
class Flauta extends InstrumentoViento {
    @Override
    public String tipoInstrumento() {
        return "Flauta";
    }
}
```

- class Flauta extends InstrumentoViento { ... }: Define una clase llamada Flauta que hereda de la clase InstrumentoViento. Esto significa que Flauta hereda todas las propiedades y métodos de InstrumentoViento.
- @Override public String tipoInstrumento() { ... }: Sobreescribe el método tipoInstrumentode la clase InstrumentoViento. Esto permite que Flauta proporcione una implementación más específica para este método (en este caso, devuelve la cadena "Flauta"). Flauta no necesita implementar los métodos tocar() y afinar() porque ya los hereda de InstrumentoViento.

EJECUTABLE

```
run:
Triángulo
Área: 6.0
Perímetro: 15.0
Cuadrilátero
Área: 16.0
Perímetro: 16.0
Tocando instrumento de viento.
Flauta
BUILD SUCCESSFUL (total time: 0 seconds)
```

Conclusiones :
Concluimos que los objetivos se cumplieron con esta practica ya que se demostró como el polimosfismo es una herramienta útil en la programación orientada a objetos que nos permite crear un codigo mas extensible , reutilizable y ordenado facil de mantener , se pudo implementar el polimorfismo de manera adecuada utilizando la herencia con clases abstractas como las interfaces.
Observamos como las clases Abstractas definen un contrato que las subclases triangulo y cuadrilátero deben cumplir , el polimorfismo nos permitió manejar objetos de diferentes clases que heredan de la misma clase abstracta y llaman a los mismo métodos área y perímetro y obtienen resultados específicos de cada uno de ls objetos.
Con el polimorfismo podemos reutilizar parte del código , escribiendo métodos que manejen objetos de la clase base o interfaz y esos mismo funcionaran en automático con cualquiera de los objetos de las subclases ademas de permitirnos tomar los detalles de cada clase y trabajar con una interfaz común que simplifica el código.

Referencias :

Awati, R. (2023, junio 19). *What is polymorphism?* TechTarget. Recuperado de https://www.techtarget.com/whatis/definition-polymorphism TechTarget

Dwivedi, A. (2021). *Object Oriented Programming Using C++ – Polymorphism* [Documento PDF]. UIET, CSJM University. Recuperado de https://gyansanchay.csjmu.ac.in/wp-content/uploads/2021/11/Object-Oriented-Programming-Using-C-Polymorphism.pdf Gyan Sanchay

"Polymorphism." (s. f.). En *Software Engineering for Self-Directed Learners*. Recuperado de https://se-education.org/se-book/oop/polymorphism/