UNIVERSIDAD TECNOLÓGICA DEL PERÚ

Facultad de Ingeniería

Programacion Orientada a Objetos

Sesión 7: Implementación de programas usando polimorfismo

Recordando:

¿Que vimos la clase pasada?

Logro de la sesión

Al finalizar la sesión, el estudiante aplica el polimorfismo en la solución de problemas e implementa soluciones que utilicen eficientemente la programación orientada a objetos en java.

Utilidad

Implementar sistemas utilizando polimorfismo no solo hace que el código sea más flexible y reutilizable, sino que también promueve buenas prácticas de diseño de software, como la abstracción, la encapsulación y la aplicación de patrones de diseño

Agenda

- Implementación de programas usando polimorfismo.

Sistema de Notificaciones

```
// Clase base abstracta Notificacion
abstract class Notificacion {
    public abstract void enviar(String mensaje);
}

// Clase derivada NotificacionEmail
class NotificacionEmail extends Notificacion {
    private String emailDestino;

    public NotificacionEmail(String emailDestino) {
        this.emailDestino = emailDestino;
    }

    @Override
    public void enviar(String mensaje) {
        System.out.println("Enviando email a " + emailDestino + ": " + mensaje);
    }
}
```

```
// Clase derivada NotificacionSMS
class NotificacionSMS extends Notificacion {
    private String numeroTelefono;

    public NotificacionSMS(String numeroTelefono) {
        this.numeroTelefono = numeroTelefono;
    }

    @Override
    public void enviar(String mensaje) {
        System.out.println("Enviando SMS a " + numeroTelefono + ": " + mensaje);
    }
}

public class TestNotificaciones {
    public static void main(String[] args) {
        Notificacion notifEmail = new NotificacionEmail("usuario@example.com");
        Notificacion notifSMS = new NotificacionSMS("1234567890");

        notifEmail.enviar("¡Hola por email!");
        notifSMS.enviar("¡Hola por SMS!");
    }
}
```

Explicación:

En este programa, la clase abstracta Notificacion tiene un método abstracto enviar(). Las clases derivadas NotificacionEmail y NotificacionSMS implementan este método para mostrar cómo se envía una notificación según el medio seleccionado.

Sistema de Vuelos Aéreos

```
// Clase base abstracta Vuelo
abstract class Vuelo {
    protected String numeroVuelo;

    public Vuelo(String numeroVuelo) {
        this.numeroVuelo = numeroVuelo;
    }

    public abstract void mostrarInformacion();
}

// Clase derivada VueloComercial
class VueloComercial extends Vuelo {
    private String aerolinea;
    private int numeroPasajeros;

    public VueloComercial(String numeroVuelo, String aerolinea, int numeroPasajeros) {
        super(numeroVuelo);
        this.aerolinea = aerolinea;
        this.numeroPasajeros = numeroPasajeros;
    }

    @Override
    public void mostrarInformacion() {
        System.out.println("Vuelo Comercial - Número: " + numeroVuelo + ", Aerolínea: "
+ aerolinea + ", Pasajeros: " + numeroPasajeros);
    }
}
```

```
// Clase derivada VueloCarga
class VueloCarga extends Vuelo {
    private double capacidadCarga;

    public VueloCarga(String numeroVuelo, double capacidadCarga) {
        super(numeroVuelo);
        this.capacidadCarga = capacidadCarga;
    }

    @Override
    public void mostrarInformacion() {
        System.out.println("Vuelo de Carga - Número: " + numeroVuelo + ", Capacidad de Carga: " +
capacidadCarga + " toneladas.");
    }
}

public class TestVuelos {
    public static void main(String[] args) {
        Vuelo vuelo1 = new VueloComercial("AA123", "American Airlines", 180);
        Vuelo vuelo2 = new VueloCarga("CARGO456", 20.5);

        vuelo1.mostrarInformacion();
        vuelo2.mostrarInformacion();
    }
}
```

Explicación:

En este programa, la clase abstracta Vuelo tiene un método abstracto mostrarInformacion(). Tenemos dos tipos de vuelos: VueloComercial, que tiene información sobre la aerolínea y el número de pasajeros, y VueloCarga, que tiene información sobre la capacidad de carga. Ambas clases derivadas implementan el método mostrarInformacion() para mostrar los detalles específicos de cada tipo de vuelo.

Practica Guiada

Implementa los ejercicios anteriormente realizados en el programa java.

Conclusiones

- Al utilizar polimorfismo, puedes reutilizar el mismo código para operar sobre diferentes tipos de objetos. Esto reduce la duplicación de código y facilita el mantenimiento del sistema, ya que los cambios solo necesitan hacerse en un lugar.

Cierre

¿Que hemos aprendido hoy?

Universidad Tecnológica del Perú

Bibliografía

- MORENO PÉREZ, J. Programación orientada a objetos.RA-MA Editorial. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=31933
- Vélez Serrano, José. Diseñar y programar, todo es empezar: una introducción a la Programación Orientada a Objetos usando UML y Java. Dykinson. https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=36368

Universidad Tecnológica del Perú