Prog. Orientada a Objetos

Carrera Programador full-stack

Polimorfismo

Polimorfismo ¿Qué es?

Polimorfismo se refiere a la capacidad de una función o método de ser utilizado de maneras diferentes en diferentes contextos en la programación. Esto significa que una función o método puede tener varias formas de ser utilizado.

Es útil porque permite crear código más flexible y reutilizable.

Polimorfismo

```
abstract class Vehiculo {
    protected velocidad : number;

     abstract avanzar(): void; // Método abstracto para polimorfismo

  }
```

La clase `Vehiculo` es la clase padre y tiene un método abstracto llamado `avanzar` no tiene una implementación.
Esto significa que cualquier clase que herede de `Vehiculo` debe sobrescribir el método `avanzar` y proporcionar su propia implementación

Polimorfismo

```
class Auto extends Vehiculo {

  public avanzar(): void {
      this.velocidad = this.velocidad + 10;
  }
}
```

```
class Motocicleta extends Vehiculo {

  public avanzar(): void {
      this.velocidad = this.velocidad + 5;
  }
}
```

Polimorfismo

Las clases `Auto` y `Motocicleta` son clases hijas que heredan de `Vehiculo` y sobrescriben el método `avanzar` de manera diferente.

La clase `Auto` tiene una implementación del método `avanzar` que suma de a 10 km, mientras que la clase `Motocicleta` tiene una implementación del método “avanzar" que suma de a 5 km.

¿Dónde se suele utilizar?

Es bastante usado en la programación, algunos ejemplos usados en la industria son:

1.El polimorfismo en un videojuego puede utilizarse para dotar a un personaje de capacidades y habilidades variadas que pueden adaptarse a contextos distintivos. Por ejemplo, un personaje puede tener la posibilidad de invocar una capacidad que puede emplearse para manifestar un resultado deseado.

2.En una aplicación de seguimiento de actividad física, se puede utilizar el polimorfismo para que diferentes tipos de actividades (como caminar, correr y hacer ejercicio en el gimnasio) se maneje el conteo de pasos de maneras diferentes.

Polimorfismo

El polimorfismo es un concepto clave en la programación que permite que una función o método tenga múltiples formas. Esto puede ser muy útil en una amplia variedad de aplicaciones, ya que permite que diferentes objetos o elementos se manejen de maneras diferentes dependiendo del contexto.

Mas ejemplos

En un sistema de autenticación de usuarios, diferentes tipos de usuarios (administradores, clientes y empleados) tienen diferentes permisos. Todos los usuarios comparten comportamientos básicos, pero algunos tienen permisos adicionales que pueden sobrescribir los métodos comunes.

```
abstract class Usuario {
    protected nombre: string;
    abstract obtenerPermisos(): void;
}

class Administrador extends Usuario {
    obtenerPermisos(): void {
        console.log(`${this.nombre} tiene todos los permisos.`);
    }
}

class Cliente extends Usuario {
    obtenerPermisos(): void {
        console.log(`${this.nombre} tiene permisos para comprar.`
);
    }
}

class Empleado extends Usuario {
    obtenerPermisos(): void {
        console.log(`${this.nombre} tiene permisos de empleado.`
);
    }
}
```

Mas ejemplos

En un sistema de e-commerce, existen diferentes formas de pago: tarjeta de crédito, Rapipago y transferencia bancaria. Todos los métodos de pago deben implementar la acción de procesar el pago, pero lo hacen de manera diferente.

```
abstract class MetodoPago {
    abstract procesarPago(monto: number): void;
}

class PagoTarjeta extends MetodoPago {
    procesarPago(monto: number): void {
        console.log(`Procesando pago de $${monto}
con tarjeta de crédito.`);
    }
}

class PagoRapipago extends MetodoPago {
    procesarPago(monto: number): void {
        console.log(`Procesando pago de $${monto}
a través de Rapipago.`);
    }
}

class PagoTransferencia extends MetodoPago {
    procesarPago(monto: number): void {
        console.log(`Procesando pago de $${monto}
mediante transferencia bancaria.`);
    }
}
```

Prog. Orientada a Objetos

Carrera Programador full-stack

Ejercicios

Ejercicio en clase

En una aplicación de mensajería o gestión de usuarios, es común enviar notificaciones a los usuarios a través de diferentes canales como correo electrónico, SMS, y notificaciones push. Cada canal tiene un proceso de envío diferente, pero todos comparten el concepto de "notificación".

- Crea una clase base llamada Notificacion con un método abstracto enviar(mensaje: string): void.
- Crea clases derivadas como Email, SMS, y PushNotification que
- sobrescriban el método enviar() para enviar el mensaje a través de
- cada canal.