

2. Polimorfismo

 Date

Empty


 Status

Not started

 Type

Empty

Units

 6. Herencia y polimorfismo

Polimorfismo

El polimorfismo es un concepto clave en la programación orientada a objetos que se refiere a la capacidad de un objeto de tomar varias formas. En otras palabras, un objeto puede tomar muchas formas diferentes y aún así ser considerado del mismo tipo.

Por ejemplo, si tenemos una clase `Animal`, podemos tener varias subclases como `Perro`, `Gato`, y `Conejo`. Cada una de estas subclases es una forma diferente de la clase `Animal`. Sin embargo, todas comparten algunas características comunes, como el hecho de que son animales y tienen ciertos comportamientos.

En Java, el polimorfismo se puede lograr a través de la herencia y las interfaces. Podemos crear una clase base, como `Animal`, y luego crear subclases como `Gato` y `Perro` que heredan de `Animal`. Luego, podemos crear una variable de tipo `Animal` y asignarle un objeto de tipo `Gato` o `Perro`. La variable se comportará como un objeto de tipo `Animal`, pero también tendrá los comportamientos específicos del `Gato` o el `Perro`.

Veamos un ejemplo en Java:

```
public class Animal { public void hacerSonido() { System.out.println("Haciendo un sonido"); } } public class Gato extends Animal { public void hacerSonido() { System.out.println("Miau"); } } public class Perro extends Animal { public void hacerSonido() { System.out.println("Guau"); } } public class Aplicacion { public static void main(String[] args) { Animal animal1 = new Gato(); Animal animal2 = new Perro(); animal1.hacerSonido(); animal2.hacerSonido(); } }
```

En este ejemplo, creamos una clase Animal con un método hacerSonido(). Luego, creamos las subclases Gato y Perro, que sobrescriben el método hacerSonido() para imprimir diferentes sonidos. En el método main(), creamos dos variables de tipo Animal y asignamos objetos de tipo Gato y Perro a ellas. Cuando llamamos al método hacerSonido() en estas variables, el método que se ejecuta depende del tipo real del objeto.

En resumen, el polimorfismo nos permite escribir código que es más genérico y reutilizable, ya que podemos tratar a diferentes objetos como si fueran del mismo tipo. Esto nos permite escribir código más flexible y escalable.

```
public class Vehiculo { public void acelerar() { System.out.p
rintln("Acelerando"); } } public class Coche extends Vehiculo
{ public void acelerar() { System.out.println("Acelerando el
coche"); } } public class Moto extends Vehiculo { public void
acelerar() { System.out.println("Acelerando la moto"); } } pu
blic class Aplicacion { public static void main(String[] arg
s) { Vehiculo vehiculo1 = new Coche(); Vehiculo vehiculo2 = n
ew Moto(); vehiculo1.acelerar(); vehiculo2.acelerar(); } }
```

En este ejemplo, creamos una clase Vehiculo con un método acelerar(). Luego, creamos las subclases Coche y Moto, que sobrescriben el método acelerar() para imprimir diferentes mensajes. En el método main(), creamos dos variables de tipo Vehiculo y asignamos objetos de tipo Coche y Moto a ellas. Cuando llamamos al método acelerar() en estas variables, el método que se ejecuta depende del tipo real del objeto.

El polimorfismo nos permite escribir código más genérico y reutilizable, ya que podemos tratar a diferentes objetos como si fueran del mismo tipo, en este caso como vehículos. Esto nos permite escribir código más flexible y escalable, y nos permite extender nuestras clases de forma sencilla.

Gracias a este concepto, si queremos tener un método que admita tanto motos como coches, tan solo tendríamos que requerir un Vehículo, entrando así el poliformismo en juego y ahorrándonos código duplicado