

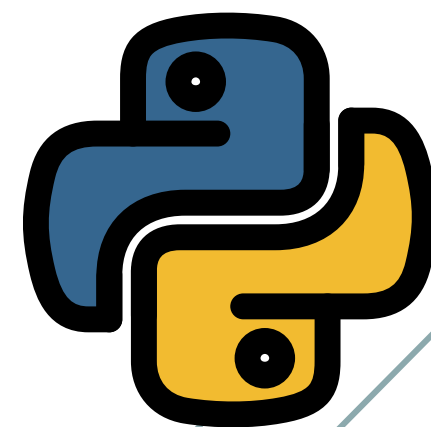
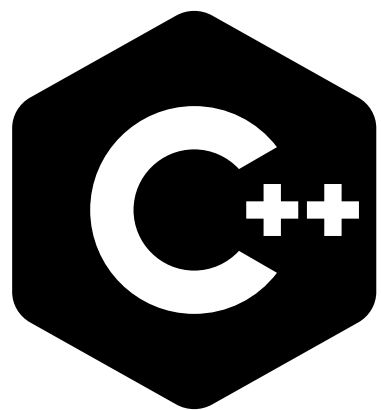


PARADIGMAS DE LA PROGRAMACION

Características de POO

941

POO



CONCEPTOS

INCLUSIÓN (SUBTIPIFICACIÓN)

Ocurre cuando una clase hereda de otra y redefine uno o más métodos de la clase base. En tiempo de ejecución, el método que se ejecuta depende del tipo del objeto en lugar del tipo de referencia.

SOBRECARGA (AD HOC)

Se refiere a la capacidad de una misma función o método para realizar diferentes tareas basadas en los tipos o número de parámetros que recibe.





SUPERCLASE

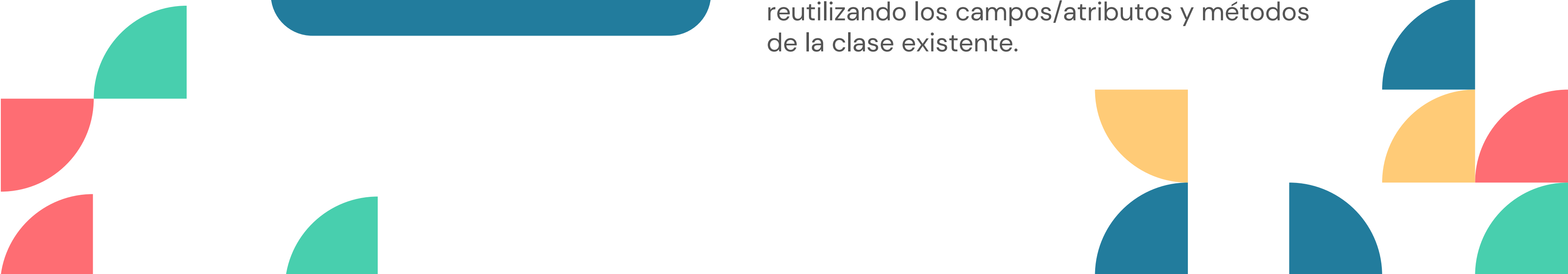
La clase cuyas características se heredan se conoce como superclase (o una clase base o una clase principal)

SUBCLASE

La clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios campos y métodos, además de los campos y métodos de la superclase.

REUTILIZACIÓN

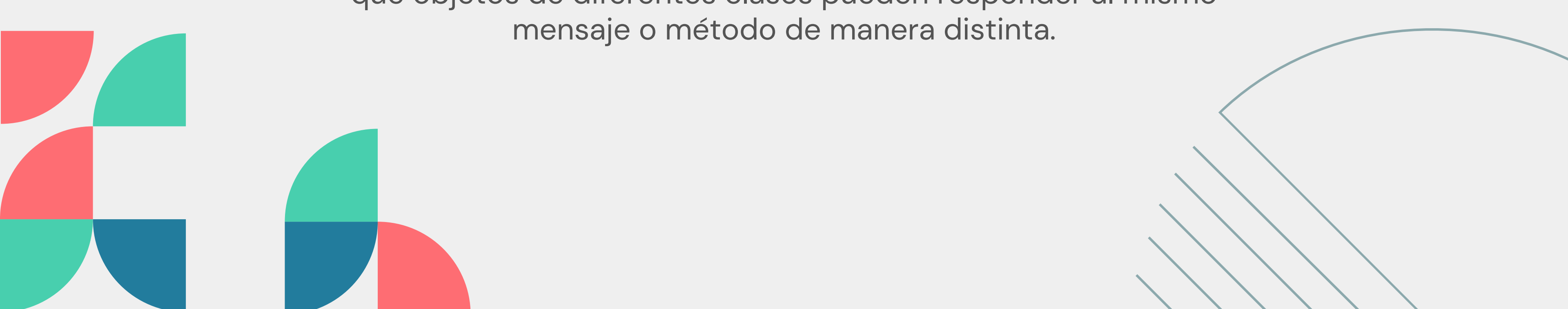
La herencia respalda el concepto de “reutilización”. Al hacer esto, estamos reutilizando los campos/atributos y métodos de la clase existente.





POLIMORFISMO

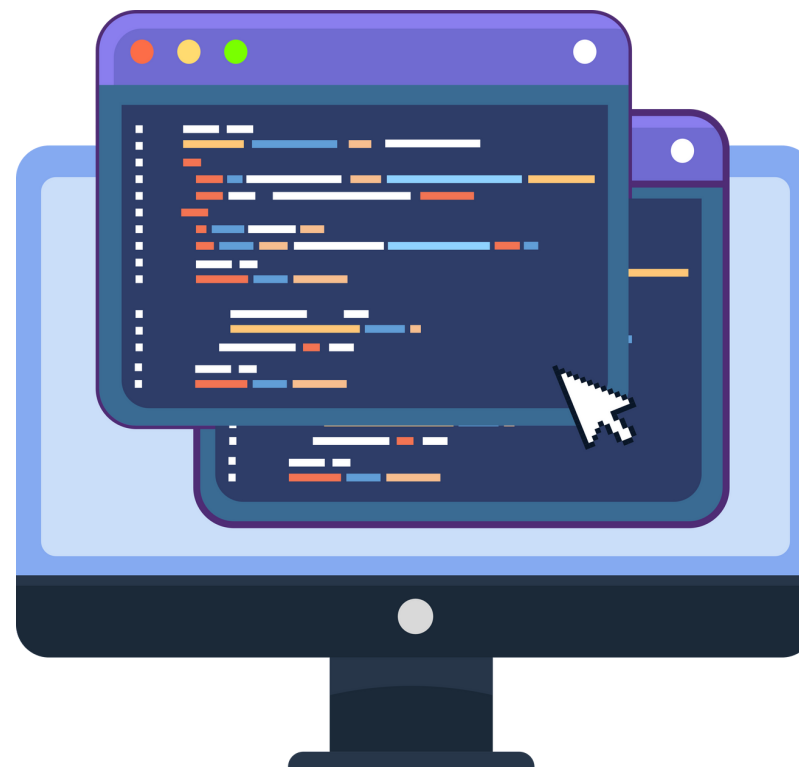
El polimorfismo es un principio mediante el cual objetos de distintas clases pueden ser tratados de manera uniforme a través de una interfaz común. En términos más simples, significa que objetos de diferentes clases pueden responder al mismo mensaje o método de manera distinta.



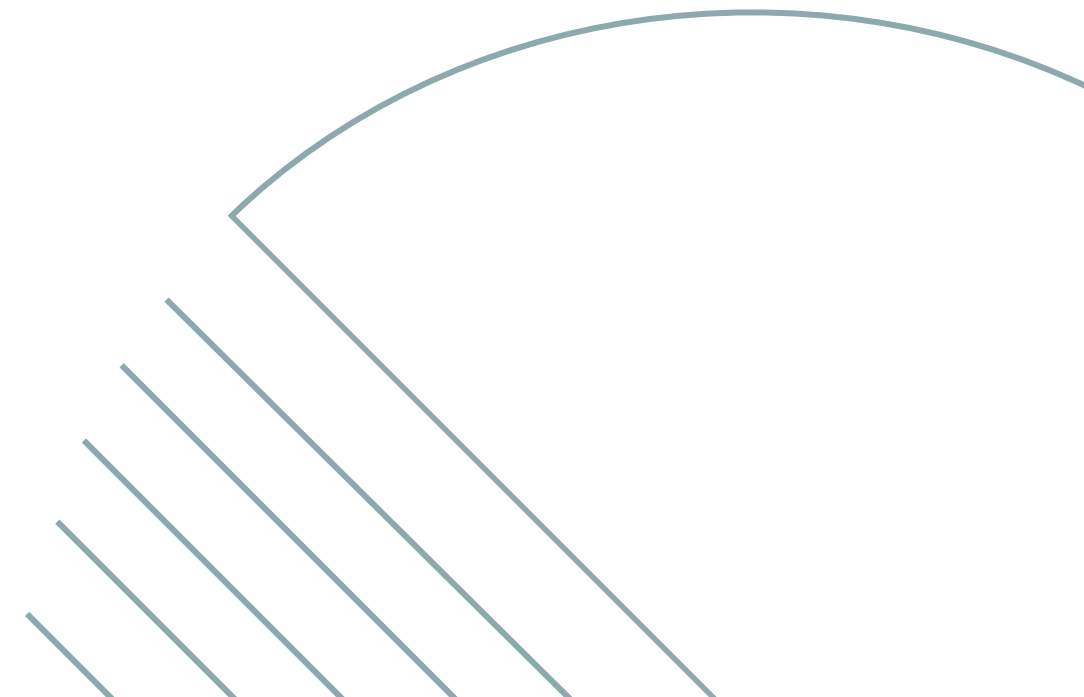
TIPOS DE POLIMORFISMO

- **Polimorfismo de sobrecarga:**
Mismo nombre de método con diferentes parámetros

- **Polimorfismo de sobrescritura:**
Un método en una subclase sobrescribe un método de la superclase



- **Polimorfismo paramétrico:**
Uso de clases genéricas que pueden trabajar con diferentes tipos de datos





GENÉRICOS

Son una característica que permite crear estructuras de datos independientes del tipo de datos. Esto significa que se pueden tener listas, colas o árboles que trabajen con cualquier tipo de datos. La funcionalidad que ofrecen estas estructuras de datos es independiente del tipo de datos que manejan.

VENTAJAS

01 - MAYOR SEGURIDAD

**02 - REDUCCION DE
CODIGO DUPLCADO**

**03 - FLEXIBILIDAD
PARA TRABAJAR
CON DIFERENTES
TIPOS DE DATOS**



EJEMPLOS

testo de ejemplo

POLIMORFISMO EN C++

La clase Animal es una clase abstracta con un método virtual puro `saludar()`, lo que significa que cualquier clase que herede de Animal debe implementar su propio método `saludar()`.

Las clases Perro y Gato heredan de Animal y proporcionan su propia implementación del método `saludar()`.

En la función `main()`, creamos instancias de Perro y Gato y llamamos a sus métodos `saludar()`, lo que demuestra el polimorfismo en acción.

```
1  #include <iostream>
2
3  // Declaración de la clase base Animal
4  class Animal {
5  public:
6      // Función virtual pura para saludar
7      virtual void saludar() const = 0;
8  };
9
10 // Clase derivada Perro
11 class Perro : public Animal {
12 public:
13     // Implementación de la función de saludo para perro
14     void saludar() const override {
15         std::cout << "¡Guau guau!" << std::endl;
16     }
17 };
18
19 // Clase derivada Gato
20 class Gato : public Animal {
21 public:
22     // Implementación de la función de saludo para gato
23     void saludar() const override {
24         std::cout << "¡Miau miau!" << std::endl;
25     }
26 };
27
28 int main() {
29     Perro perro;
30     Gato gato;
31
32     // Polimorfismo: Llamando al método saludar de cada objeto
33     std::cout << "El perro dice: ";
34     perro.saludar();
35
36     std::cout << "El gato dice: ";
37     gato.saludar();
38
39     return 0;
40 }
```

GENÉRICOS EJEMPLO: PYTHON

En Python, los genéricos se implementan mediante el uso de clases genéricas y decoradores.

PILAS

Este código define una clase genérica de cola en Python utilizando el módulo collections y el módulo typing.

La clase Queue puede trabajar con cualquier tipo de datos, ya que el tipo de elementos que se almacenan en la cola se especifica mediante el parámetro de tipo T. La cola se implementa utilizando una cola deque de la biblioteca estándar de Python. La clase Queue tiene dos métodos: push para agregar un elemento al final de la cola y pop para eliminar y devolver el primer elemento de la cola.

```
1  # generic_queue.py
2
3  from collections import deque
4  from typing import Generic, TypeVar
5
6  T = TypeVar("T")
7
8  class Queue(Generic[T]):
9      def __init__(self) -> None:
10         self.elements: deque[T] = deque()
11
12         def push(self, element: T) -> None:
13             self.elements.append(element)
14
15         def pop(self) -> T:
16             return self.elements.popleft()
```

GENÉRICOS EJEMPLO: PYTHON

En Python, los genéricos se implementan mediante el uso de clases genéricas y decoradores.

LISTA

En este código, se está utilizando una cola genérica definida en el módulo `generic_queue`.

La cola puede almacenar elementos de cualquier tipo, pero en este caso, se está creando una cola específica para enteros `Queue[int]()`. Se agregan dos enteros a la cola usando el método `push`, y se puede ver el contenido de la cola usando la variable `elements`.

La cola está implementada usando una cola doblemente terminada `deque` de la biblioteca estándar de Python. Finalmente, se elimina y devuelve el primer elemento de la cola usando el método `pop`. El valor devuelto es el primer elemento que se agregó a la cola, en este caso, el valor 3.

```
1  from generic_queue import Queue
2  queue = Queue[int]()
3
4  queue.push(3)
5  queue.push(12)
6  queue.elements
7  deque([3, 12])
8
9  queue.pop()
10 3
```

GENÉRICOS EJEMPLO: PYTHON

En Python, los genéricos se implementan mediante el uso de clases genéricas y decoradores.

COLAS

En este ejemplo, T es un parámetro de tipo que se puede reemplazar con cualquier tipo de datos al llamar a la función. La función search toma tres parámetros: array es una lista de elementos del tipo T, item es el elemento que se busca, y compare es una función de comparación que toma dos parámetros del tipo T y devuelve un valor booleano.

La función search itera sobre la lista y compara cada elemento con el elemento buscado utilizando la función de comparación. Si se encuentra un elemento que coincide con el elemento buscado, la función devuelve True, de lo contrario, devuelve False.

```
python
from typing import TypeVar, List

T = TypeVar('T')

class Queue(List[T]):
    def enqueue(self, item: T) -> None:
        self.append(item)

    def dequeue(self) -> T:
        return self.pop(0)
```

**GRACIAS POR SU
ATENCIÓN PAPUS:V**

