Bridge

Desacopla una abstracción de su implementación, para que las dos puedan cambiar de forma independiente

Abstracción

Representa la parte abstracta de un objeto y define la interfaz con la que interactúan los clientes.

Clase Refinada

Crea una subclase de la clase de abstracción que refina las operaciones definidas en la abstracción utilizando las operaciones de implementación.

Implementación

Define una interfaz que representa la implementación concreta que será utilizada por la abstracción. Los métodos en la interfaz de implementación son los que realizarán las operaciones concretas.

Implementaciones Concretas

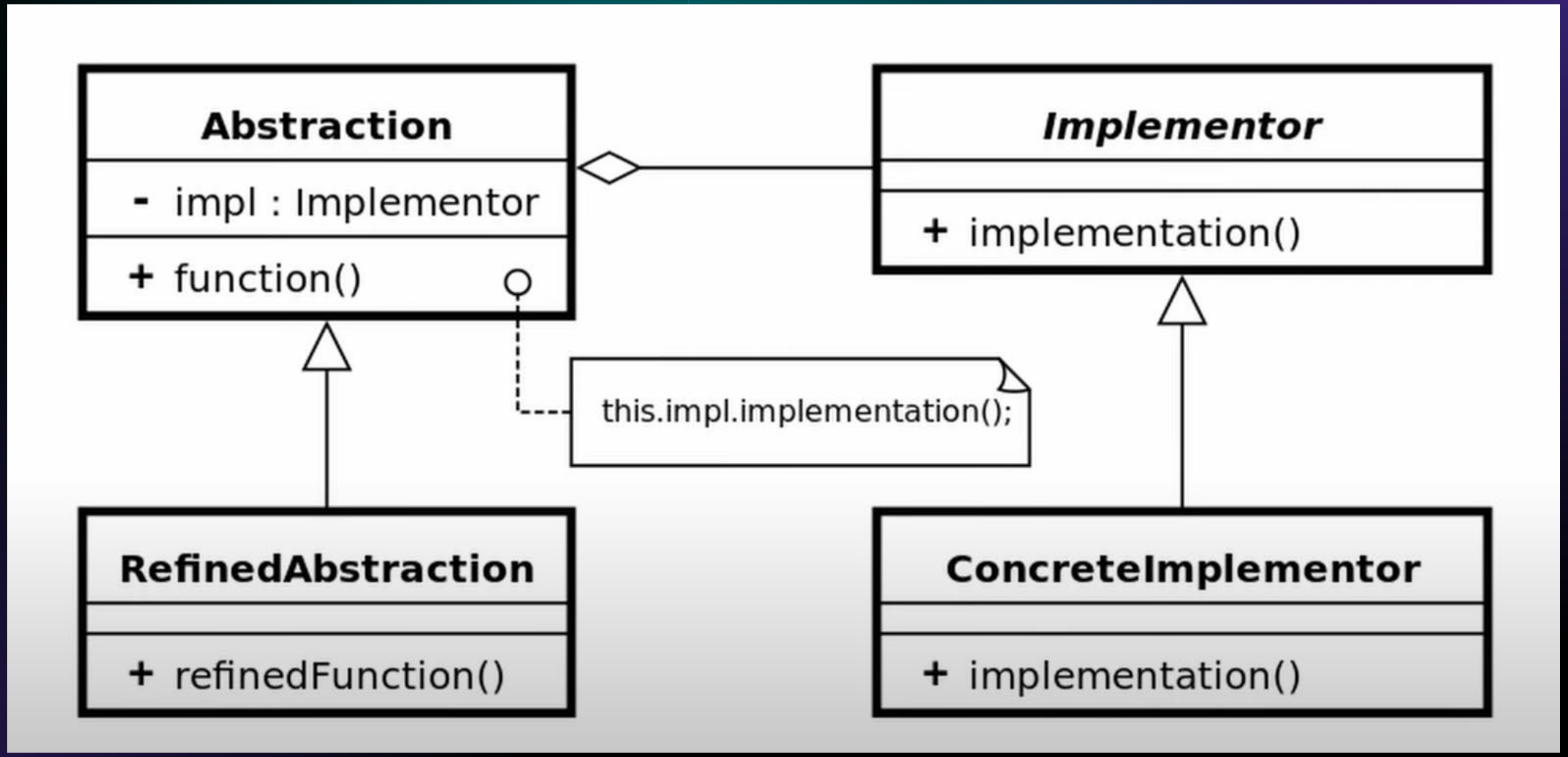
Cada implementación concreta proporciona una implementación específica de las operaciones definidas en la interfaz de implementación.

```python
# Implementaciones Concretas - Formas Geométricas con Colores
class RedCircle:
    def apply_color(self):
        return "Círculo de color rojo"

class GreenCircle:
    def apply_color(self):
        return "Círculo de color verde"

class RedSquare:
    def apply_color(self):
        return "Cuadrado de color rojo"

class GreenSquare:
    def apply_color(self):
        return "Cuadrado de color verde"

# Uso de las Implementaciones Concretas
red_circle = RedCircle()
green_circle = GreenCircle()
red_square = RedSquare()
green_square = GreenSquare()

print(red_circle.apply_color())    # Imprime "Círculo de color rojo"
print(green_circle.apply_color())  # Imprime "Círculo de color verde"
print(red_square.apply_color())    # Imprime "Cuadrado de color rojo"
print(green_square.apply_color())  # Imprime "Cuadrado de color verde"
```

```python
# Abstracción - Forma
class Shape:
    def __init__(self, color):
        self.color = color

    def apply_color(self):
        pass

# Implementación - Color
class Color:
    def apply_color(self):
        pass

# Implementaciones Concretas - Colores Concretos
class RedColor(Color):
    def apply_color(self):
        return "Rojo"

class GreenColor(Color):
    def apply_color(self):
        return "Verde"

# Abstraccion Refinada - Formas Concretas
class Circle(Shape):
    def apply_color(self):
        return f"Círculo de color {self.color.apply_color()}"

class Square(Shape):
    def apply_color(self):
        return f"Cuadrado de color {self.color.apply_color()}"

```

```python
# Uso del patrón Bridge
red = RedColor()
green = GreenColor()

circle = Circle(red)
square = Square(green)

print(circle.apply_color())  # Imprime "Círculo de color Rojo"
print(square.apply_color())  # Imprime "Cuadrado de color Verde"
```

Relación de pertenencia



Beneficios del patrón Bridge:

- Desacopla las abstracciones de las implementaciones, lo que facilita la extensibilidad y la variabilidad independiente.
- Permite cambiar o extender las implementaciones sin afectar a las abstracciones existentes y viceversa.
- Facilita la composición en lugar de la herencia, evitando la creación de una jerarquía de clases excesivamente compleja

- Gracias