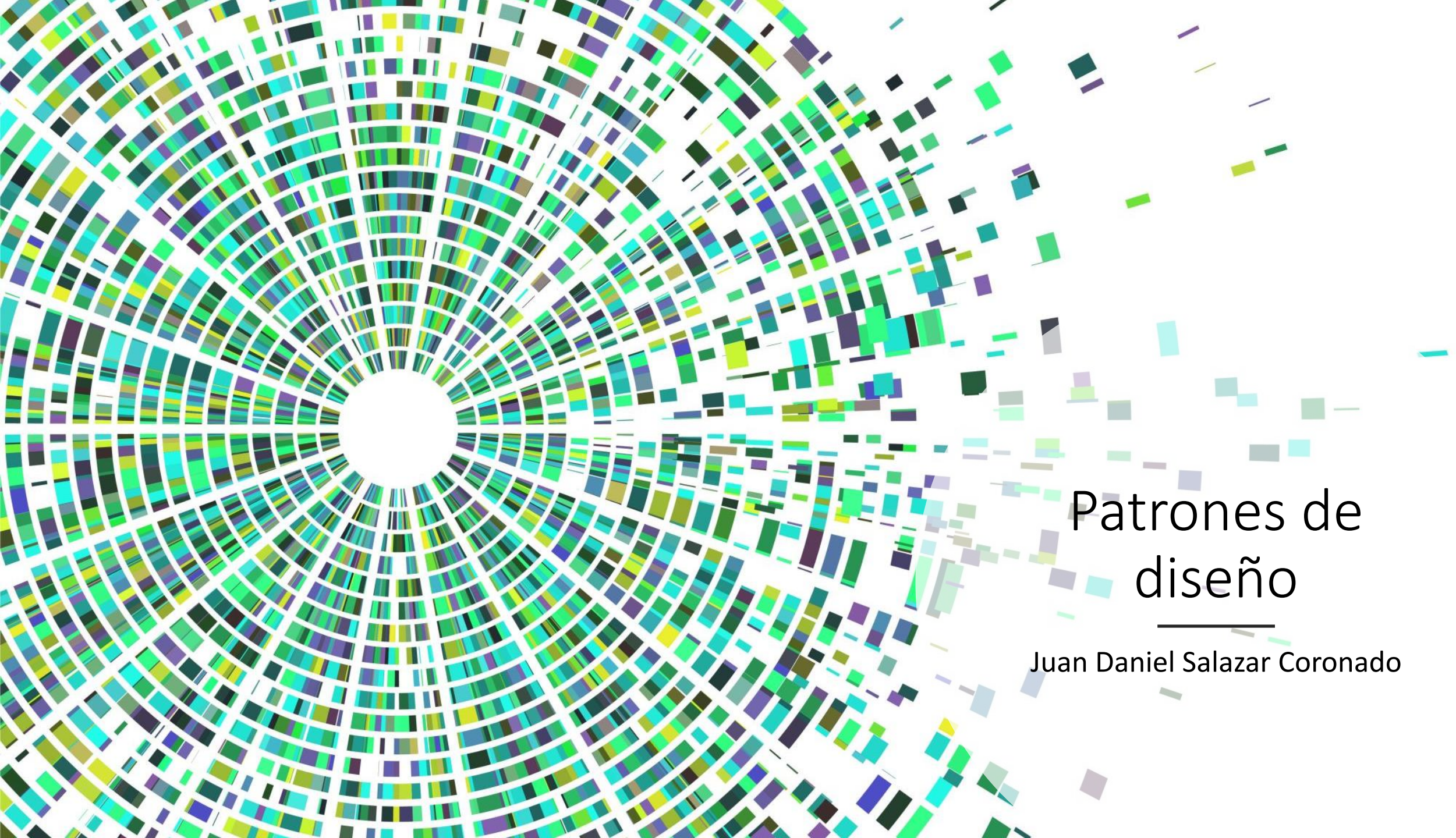


Patrones de diseño

Juan Daniel Salazar Coronado

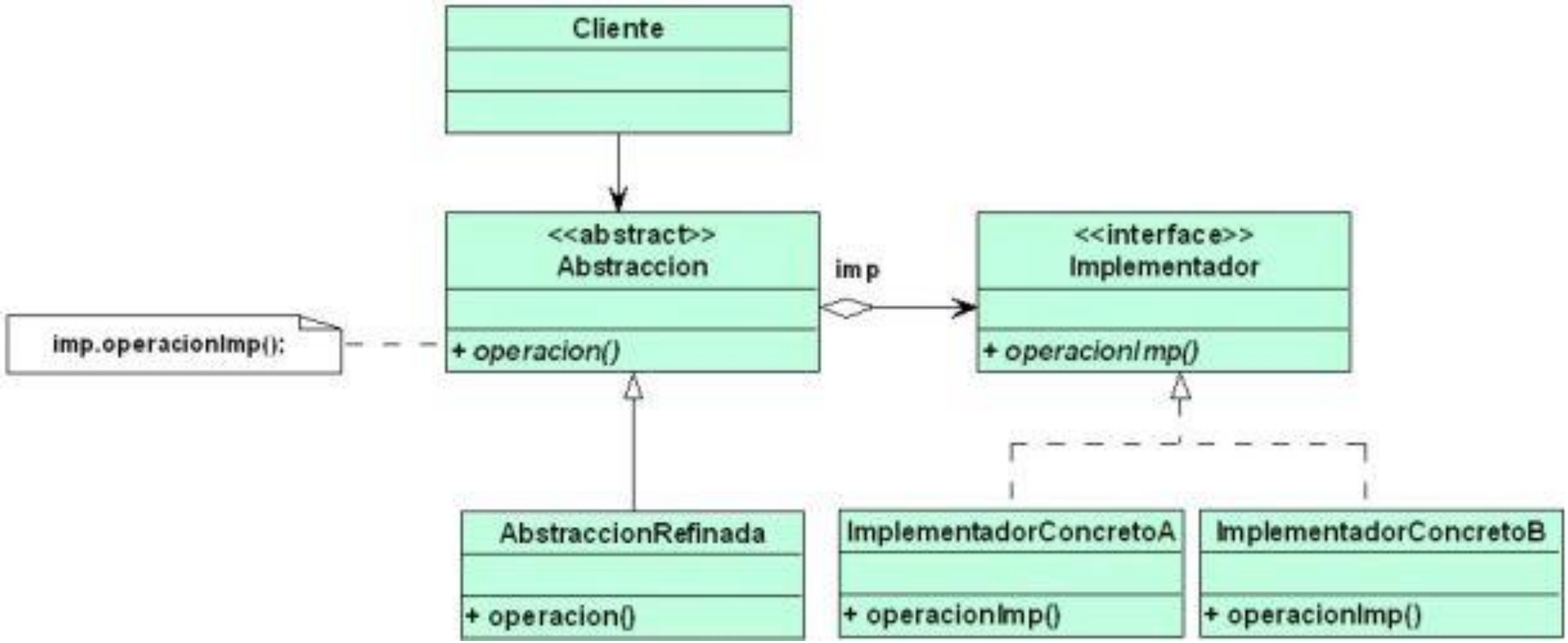
Bridge

ESTÁ EN LA CLASIFICACIÓN DE PATRONES ESTRUCTURALES

ES UN PATRÓN DE DISEÑO ESTRUCTURAL QUE TE PERMITE DIVIDIR UNA CLASE GRANDE O VARIAS CLASES RELACIONADAS EN DOS JERARQUÍAS SEPARADAS, QUE PUEDEN DESARROLLARSE SIN DEPENDER DE LA OTRA.

TAMBIÉN ES CONOCIDO COMO HANDLE/BODY

Estructura de Bridge



Participantes

- Cliente: Como siempre, se encargará de utilizar las abstracción de la que trata el problema.
- Abstraccion: Abstracción objetivo del problema. Contiene una referencia a un objeto que implemente la interfaz Implementator el cual servirá para definir la funcionalidad de la misma. Puede ser una clase abstracta o una interfaz.
- Implementador: Interfaz que define las operaciones necesarias para implemntar la funcionalidad de Abstraction.
- AbstracciónRefinada: Clase que hereda o implementa (según sea clase abstracta o interfaz respectivamente) de Abstraction. Extiende su funcionalidad basándose en el atributo que almacena al Implementator.
- ImplementadorConcreto: Implementación concreta de la funcionalidad definida por el Implementor. Puede haber varias implementaciones para la misma funcionalidad.

Ejemplo de código

```python
class Abstraction:
    def __init__(self, imp):
        self._imp = imp

    def operation(self, a, b):
        self._a = a
        self._b = b
        print("Haciendo uso de la abstracción normal\n"
              f"{self._imp.operation_imp(self._a, self._b)}")

class ExtendedAbstraction(Abstraction):
    def operation(self, a, b):
        self._a = a
        self._b = b
        print("Haciendo uso de la abstracción extendida\n"
              f"{self._imp.operation_imp(self._a, self._b)}")

class Implementor():
    def operation_imp(self, a, b):
        pass

class suma(Implementor):
    def operation_imp(self, a, b):
        self.resultado = a + b
        return "Suma de {} y {} = {}".format(a, b, self.resultado)
```

```python
class resta(Implementor):
    def operation_imp(self, a, b):
        self.resultado = a - b
        return "Resta de {} y {} = {}".format(a, b, self.resultado)

class division(Implementor):
    def operation_imp(self, a, b):
        self.resultado = a / b
        return "División de {} y {} = {}".format(a, b, self.resultado)

class multiplicacion(Implementor):
    def operation_imp(self, a, b):
        self.resultado = a * b
        return "Multiplicación de {} y {} = {}".format(a, b, self.resultado)


def main():
    unasuma = suma()
    abstraction = Abstraction(unasuma)
    abstraction.operation(float(5), float(4))

    abstractionext = ExtendedAbstraction(unasuma)
    abstractionext.operation(float(6), float(7))
```

```python
if __name__ == "__main__":
    main()
```

```
Haciendo uso de la abstracción normal
Suma de 5.0 y 4.0 = 9.0
Haciendo uso de la abstracción extendida
Suma de 6.0 y 7.0 = 13.0
```

Aplicaciones y usos conocidos

Este patrón se usa cuando:

- Se desea evitar un enlace permanente entre la abstracción y su implementación. Esto puede ser debido a que la implementación debe ser seleccionada o cambiada en tiempo de ejecución.
- Para compartir una implementación entre múltiples objetos, sin que lo noten los clientes.

Usos conocidos:

• El patrón Bridge se utiliza ampliamente en los sistemas gráficos de ventanas, así como aplicación de las bibliotecas y kits de desarrollo.