Patrones de diseño 3

DECORATOR PATTERN

DECORATOR PATTERN

- **Intención:**
  - Puede ser usado para extender funcionalidad de un cierto objeto. De tal forma que se puede formar un stack de funcionalidades
- **Motivación:**
  - Cuando quieres extender una clase base de tal forma que puedas hacer un stack de llamadas para agregar o modificar funcionalidad.

StarBuds

Starbuds Sistema de calculo de precio

Starbuds Menu:

Cafes:

Café Chiapas:	1.5
Café Americano:	1.0

Toppings:

Leche:	1.0
Leche Dietética:	1.5
Doble shot de café:	0.3
Chocolate:	0.3
Crema batida:	0.7

Starbuds Sistema de calculo de precio

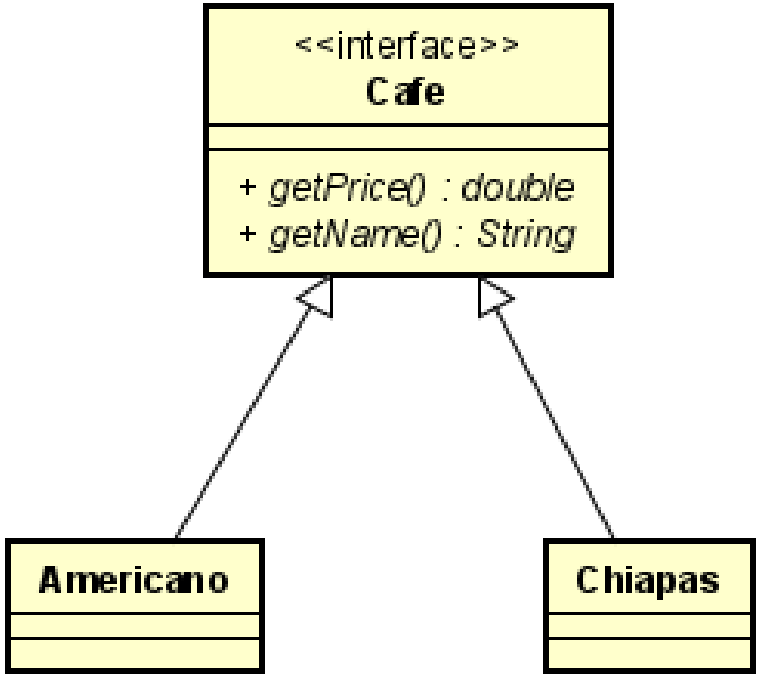
Starbuds Menu:

<u>Cafes</u>:	
Café Chiapas:	1.5
Café Americano:	1.0
<u>Toppings</u>:	
Leche:	1.0
Leche Dietética:	1.5
Doble shot de café:	0.3
Chocolate:	0.3
Crema batida:	0.7

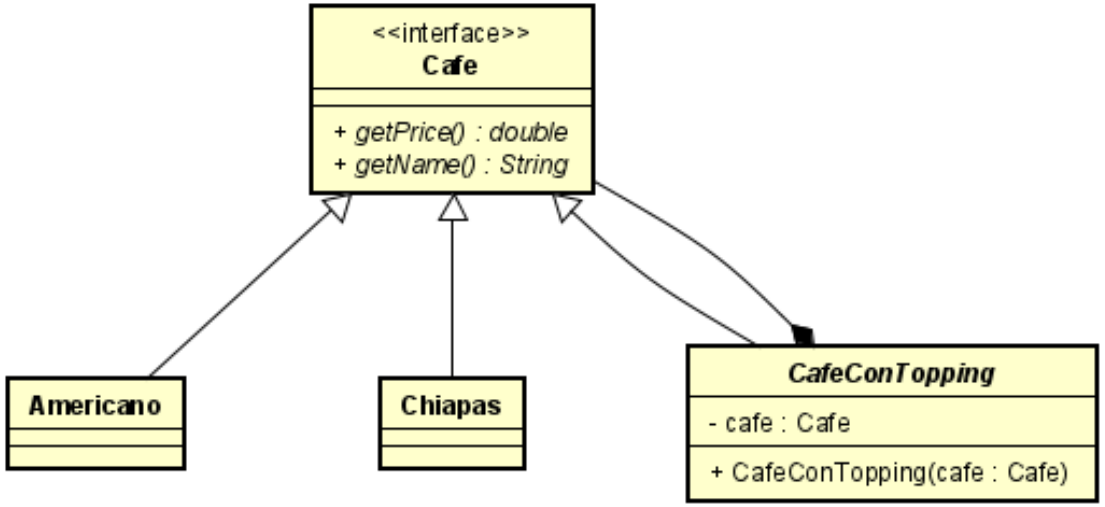
Karen:
Quiero un doble shot americano con leche dietética y un poco de chocolate

```
class Cafe {
    getPrice(cafe: string, toppings: string[]) {
        let price :number = 0;
        switch(cafe){
            case 'american':
                price += 1;
            case 'chiiapas':
                price += 1.5;
        }


        if (toppings.contains('milk')) {
            price += 1.0;
        }

        if (toppings.contains('diet milk')) {
            price += 1.5;
        }

        //....

        if (toppings.contains('doble shot')) {
            price += 0.3;
        }
    }
}
```

Starbuds Sistema de calculo de precio

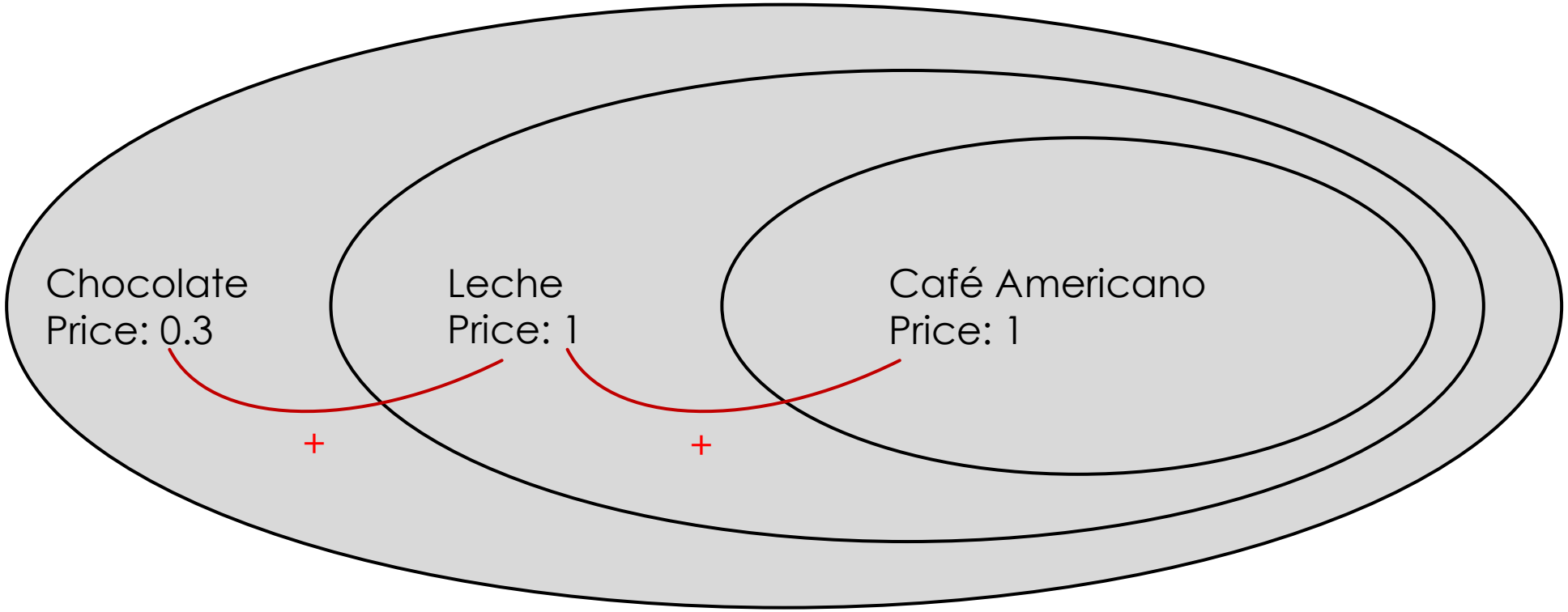


Starbuds implementación de cafés



Implementación de los toppings

Creando stack de funcionalidades



```
export default interface ICoffee {
    printName(): string;
    getPrice(): number;
}
```

Interfaz para Cafés

```
export default class AmericanCoffee implements ICoffee {
    printName(): string {
        return "Coffee";
    }

    getPrice(): number {
        return 1;
    }
}
```

Clase para un café en concreto

Interfaz para los toppings

```
export default abstract class CoffeeWithTopping implements ICoffee {

    constructor(
        protected coffee: ICoffee
    ) {}

    abstract printName(): string;
    abstract getPrice(): number;
}
```

```
export default class Milk extends CoffeeWithTopping {
    printName(): string {
        return `${this.coffee.printName()} / with milk`;
    }

    getPrice(): number {
        return this.coffee.getPrice() + 1;
    }
}
```

Clase concreta de un topping

Interacción entre clases

```
export default interface ICoffee {
    printName(): string;
    getPrice(): number;
}
```

```
export default abstract class CoffeeWithTopping implements ICoffee {

    constructor(
        protected coffee: ICoffee
    ) {}

    abstract printName(): string;
    abstract getPrice(): number;
}
```

```
export default class AmericanCoffee implements ICoffee {
    printName(): string {
        return "Coffee";
    }

    getPrice(): number {
        return 1;
    }
}
```

```
export default class Milk extends CoffeeWithTopping {
    printName(): string {
        return `${this.coffee.printName()} / with milk`;
    }

    getPrice(): number {
        return this.coffee.getPrice() + 1;
    }
}
```

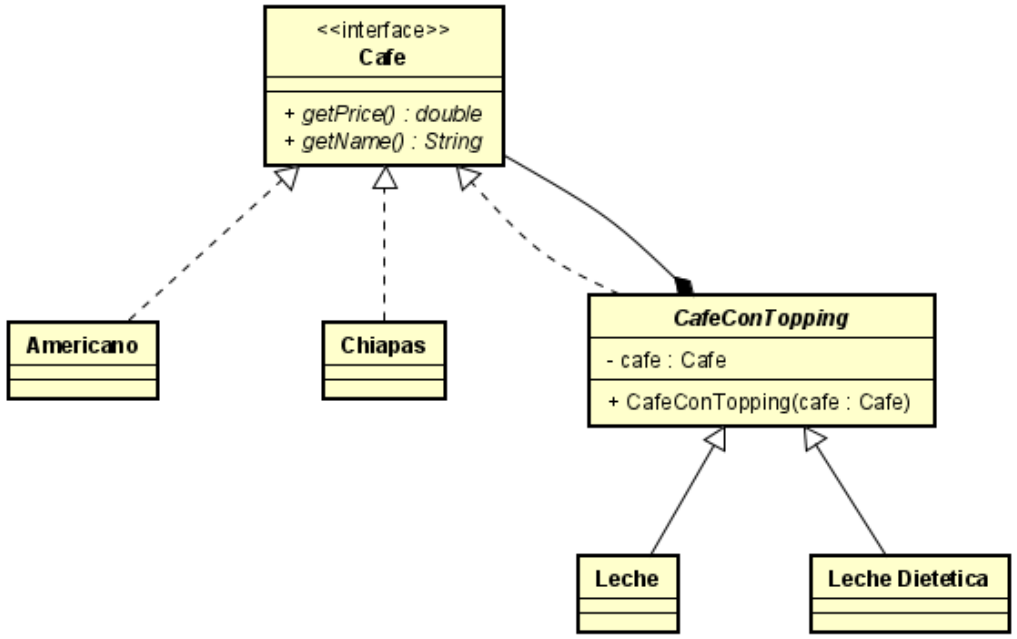
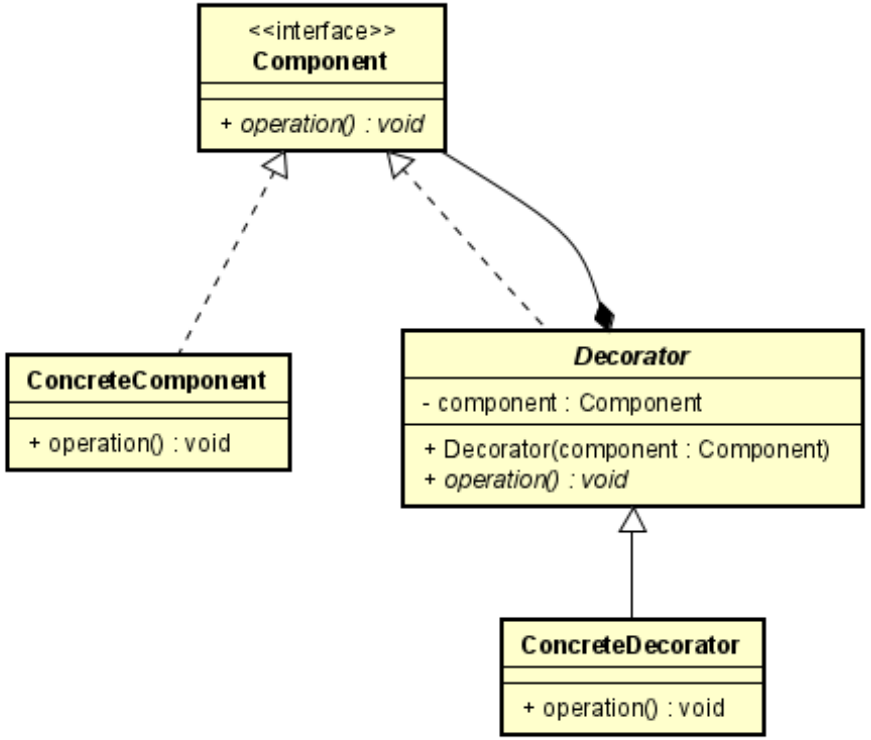
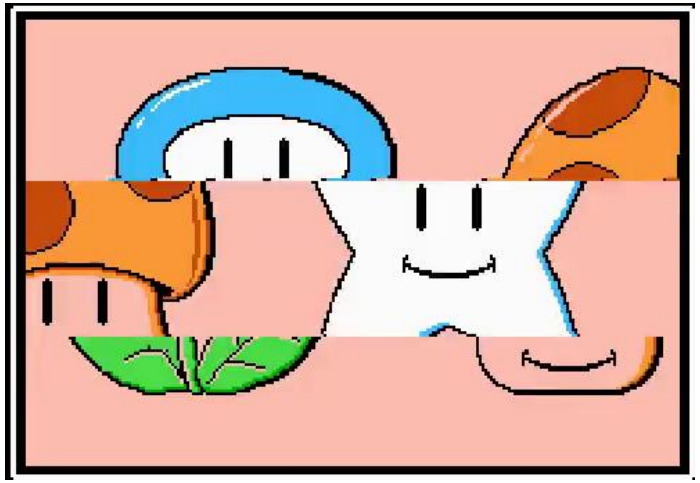


Diagrama de clases

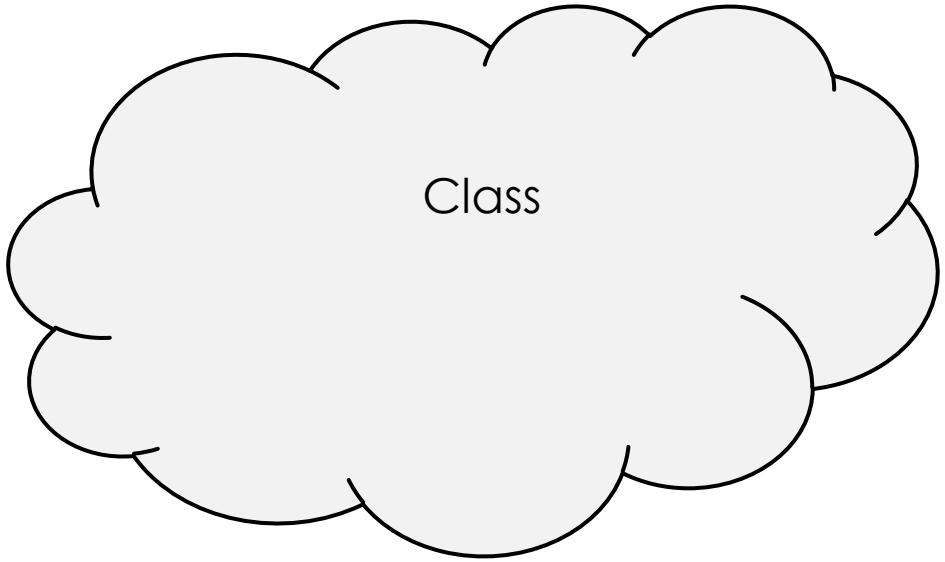


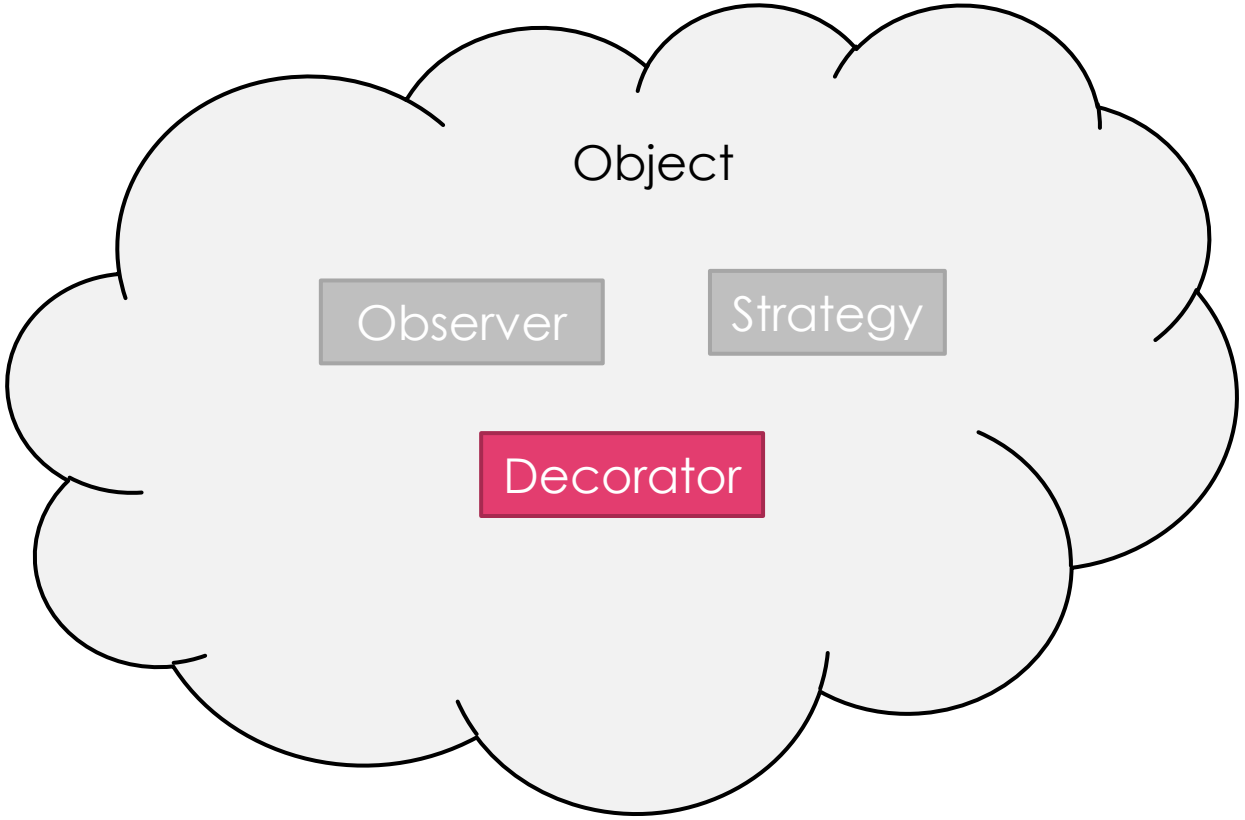
El patron decorator



Decorator VS Strategy

Categorías de patrones
Objects - Classes





Categorías de patrones
Creational – Behavioral - Structural