Patrón Strategy

PATRÓN DE COMPORTAMIENTO

Objetivo

El objetivo del **patrón Strategy** es tener diferentes algoritmos o **diferentes lógicas encapsuladas**, para así tener la capacidad de poder **cambiar** de una a otra de una forma sencilla y muy elegante.

Una batalla Pokémon

Ataques

Los **ataques** pueden causar **daño** al Pokémon enemigo o al **entorno de la batalla.**

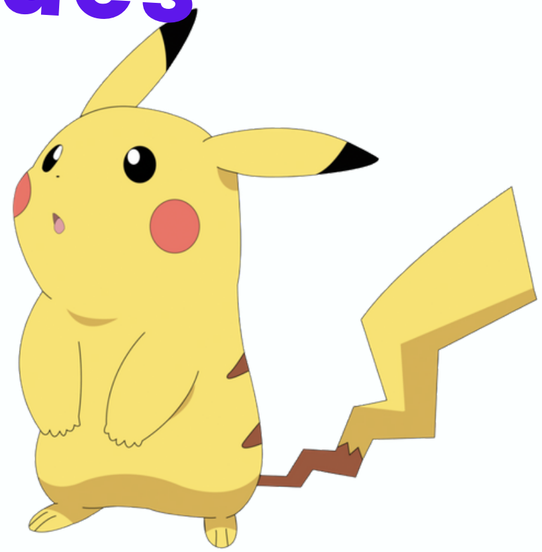
Las variantes de ataque son demasiadas

```
function computePokemonAttack(battle: BattleState, selectedAttack: Attack): BattleState {
  //Damage functions
  if (attack.type == 'PHYSICAL') {
   battle.foePokemon.makePhysicalDamage();
  }
  else if (attack.type == 'SPECIAL') {
   battle.foePokemon.makeSpecialDamage();
  }
  //Status change functions
  if (attack.diminishesPhysicalAttackStat) {
   battle.foePokemon.diminishPhysicalAttack();
  }
  if (attack.diminishesSpecialAttackStat) {
   battle.foePokemon.diminishSpecialAttack();
  }
  if (attack.diminishesSpeedStat) {
   battle.foePokemon.diminishSpeed();
  }
  if (attack.diminishesPhysicalDefenseStat) {
   battle.foePokemon.diminishPhysicalDefense();
  }
  if (attack.diminishesSpecialDefenseStat) {
   battle.foePokemon.diminishSpecialDefense();
  }

  //Battle settings functions
  if (attack.canMakeRain) {
    if (Math.random() > 0.5) {
      battle.startRain();
    }
  }
  if (attack.canMakeSandStorm) {
    if (Math.random() > 0.5) {
      battle.startSandStorm();
    }
  }

  return battle;
}
```

Esta solución se nos complica rápidamente...

+700 Ataques

Name	Type	Cat.	Power	Acc.	PP	TM	Effect	
10,000,000 Volt Thunderbolt	ELECTRIC		195	—	1		Pikachu-exclusive Z-Move. High critical hit ratio.	—
Absorb	GRASS		20	100	25		User recovers half the HP inflicted on opponent.	—
Accelerock	ROCK		40	100	20		User attacks first.	—
Acid	POISON		40	100	30		May lower opponent's Special Defense.	10
Acid Armor	POISON		—	—	20		Sharply raises user's Defense.	—
Acid Downpour	POISON	—	—	—	1		Poison type Z-Move.	—
Acid Spray	POISON		40	100	20		Sharply lowers opponent's Special Defense.	100
Acrobatics	FLYING		55	100	15	TM78	Stronger when the user does not have a held item.	—
Acupressure	NORMAL		—	—	30		Sharply raises a random stat.	—
Aerial Ace	FLYING		60	∞	20		Ignores Accuracy and Evasiveness.	—
Aeroblast	FLYING		100	95	5		High critical hit ratio.	—
After You	NORMAL		—	—	15		Gives target priority in the next turn.	—
Agility	PSYCHIC		—	—	30		Sharply raises user's Speed.	—
Air Cutter	FLYING		60	95	25		High critical hit ratio.	—
Air Slash	FLYING		75	95	15	TM95	May cause flinching.	30
All-Out Pummeling	FIGHTING	—	—	—	1		Fighting type Z-Move.	—
Ally Switch	PSYCHIC		—	—	15		User switches with opposite teammate.	—



¿Cómo programar esto sin que se vuelva un caos?

SIN IF..ELSE NI SWITCH CASE

Strategy: Pokémon

```
function computeTurn (state: BattleState,
                      myAttack: AttackStrategy,
                      foeAttack: AttackStrategy) {
    //  ...
    // Some logic of the turn computation
    // Mi pokémon siempre ataca primero aqui 😇
    state = myAttack.ejecutarAtaque(state);
    state = foeAttack.ejecutarAtaque(state);
    // Some more logic for the turn computation
    //  ...
}
```

Strategy: Pokémon

```
function computeTurn (state: BattleState,
                      myAttack: AttackStrategy,
                      foeAttack: AttackStrategy) {
    //  ...
    // Some logic of the turn computation
    // Mi pokémon siempre ataca primero aqui 😇
    state = myAttack.ejecutarAtaque(state);
    state = foeAttack.ejecutarAtaque(state);
    // Some more logic for the turn computation
    //  ...
}
```

```
export interface AttackStrategy {
    ejecutarAtaque(battle: BattleState): BattleState
}
```

Strategy: Pokémon

```
function computeTurn (state: BattleState,
                      myAttack: AttackStrategy,
                      foeAttack: AttackStrategy) {
    //  ...
    // Some logic of the turn computation
    // Mi pokémon siempre ataca primero aqui 😇
    state = myAttack.ejecutarAtaque(state);
    state = foeAttack.ejecutarAtaque(state);
    // Some more logic for the turn computation
    //  ...
}
```

```
export interface AttackStrategy {
    ejecutarAtaque(battle: BattleState): BattleState
}
```

```
export class RainStrategy implements AttackStrategy {
    ejecutarAtaque(battle: BattleState): BattleState {
        let randValue = Math.random();
        if (randValue > 0.5) {
          battle.startRain(); 🌧🌧🌧🌧🌧🌧
        }
        return battle;
    }
}
```

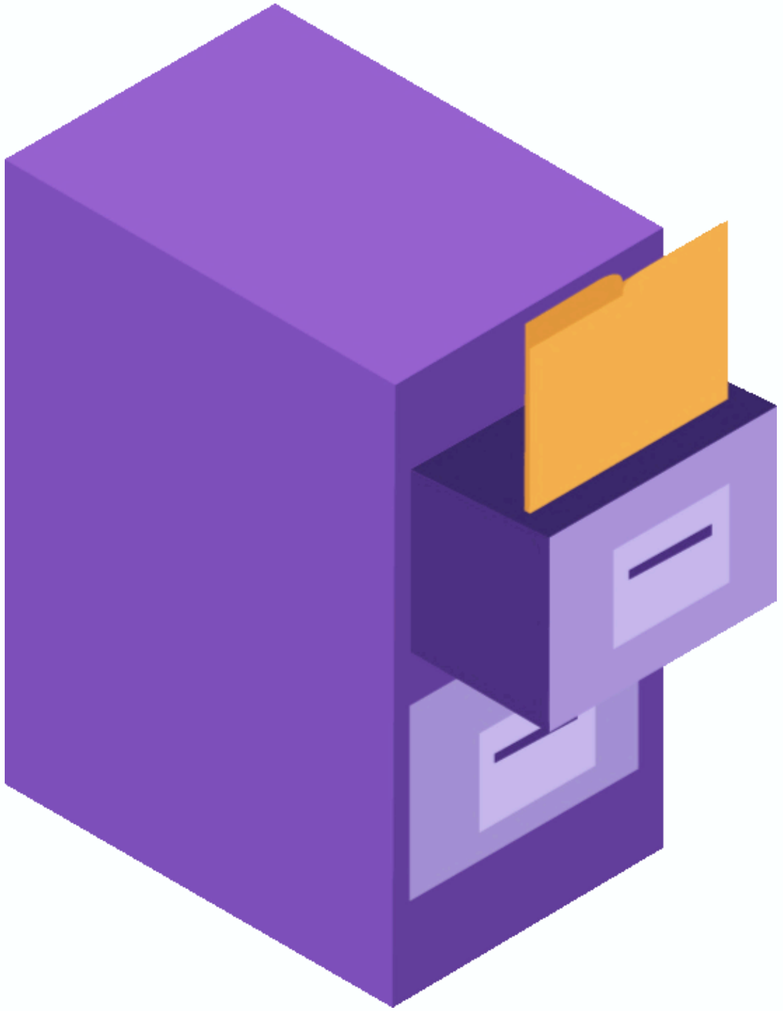
Strategy: Pokémon

```
function computeTurn (state: BattleState,
                      myAttack: AttackStrategy,
                      foeAttack: AttackStrategy) {
    // ...
    // Some logic of the turn computation
    // Mi pokémon siempre ataca primero aqui 😇
    state = myAttack.ejecutarAtaque(state);
    state = foeAttack.ejecutarAtaque(state);
    // Some more logic for the turn computation
    // ...
}
```

```
export interface AttackStrategy {
    ejecutarAtaque(battle: BattleState): BattleState
}
```

```
export class DiminishStatStrategy implements AttackStrategy {
    private stat: PokemonStat;

    constructor(stat: PokemonStat) {
        // Podemos usar constructores para personalizar estrategias
        this.stat = stat;
    }

    ejecutarAtaque(battle: BattleState): BattleState {
        battle.foePokemon.diminishStat(this.stat);
        return battle;
    }
}
```

Otro ejemplo: Logger



Strategy: Logger

```typescript
export interface Logger: void {
    write(message: string): void
}
```

```typescript
export class ConsoleLogger implements Logger {
    write(message: string): void {
        console.log(message);
    }
}
```

```typescript
const fs = require('fs');

export class FileLogger implements Logger {
    private filePath: string;

    constructor(filePath: string) {
        this.filePath = filePath;
    }
    write(message: string): void {
        //Logic to write to file
        fs.appendFileSync(this.filePath, message);
    }
}
```

Strategy: Logger

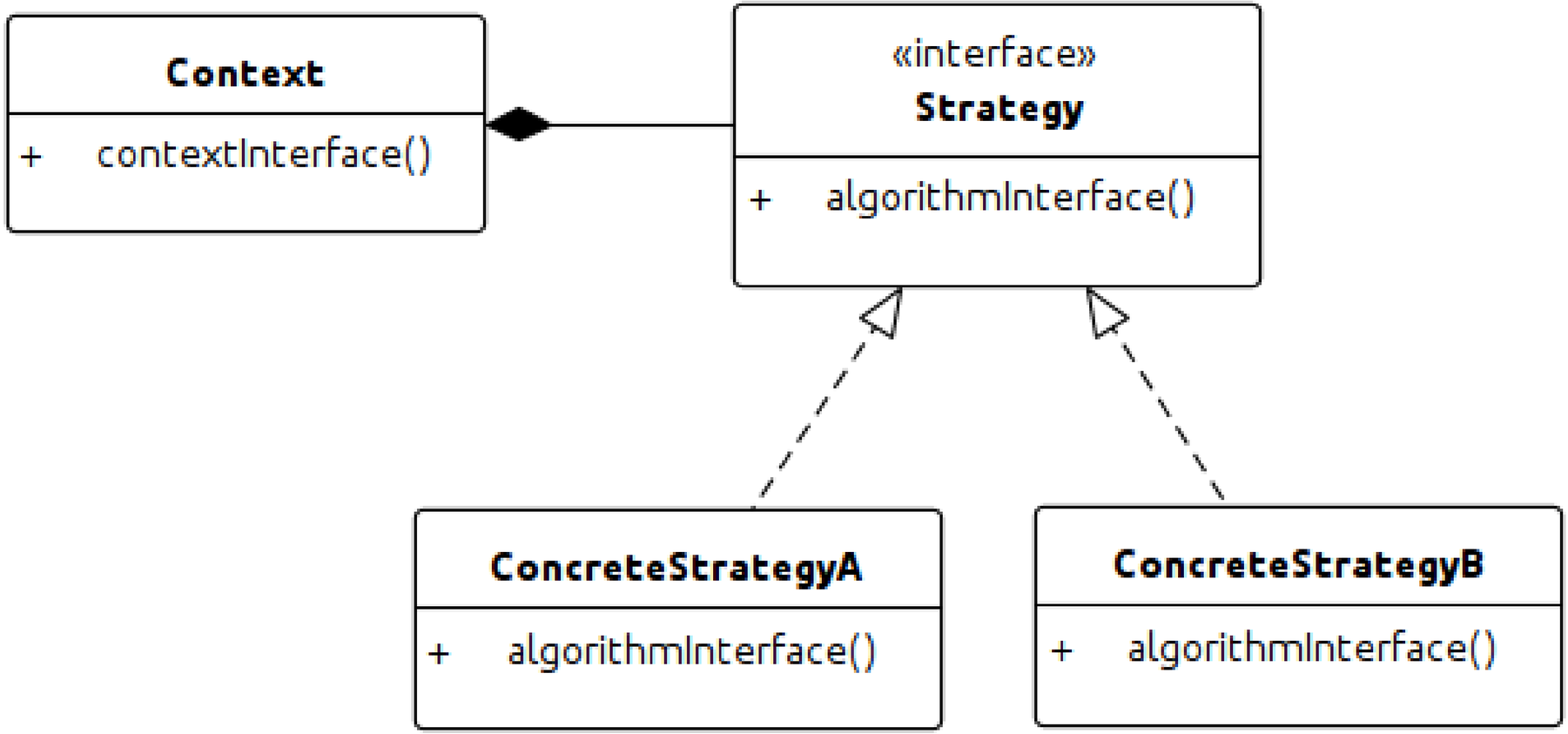
```
export class App {
    run(logger: Logger): void {
        //Do things, draw some asterisks
        for(let i=0; i=4; ++i){
            logger.write("*".repeat(i));
        }
    }
}
```

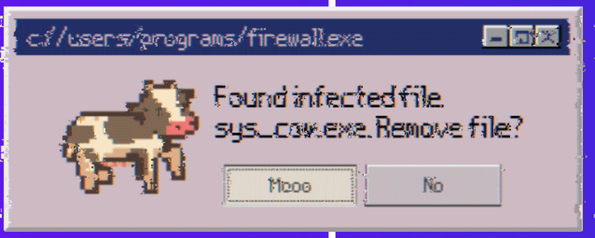
```
export interface Logger: void {
    write(message: string): void
}
```

```
export class ConsoleLogger implements Logger {
    write(message: string): void {
        console.log(message);
    }
}
```

```
const fs = require('fs');

export class FileLogger implements Logger {
    private filePath: string;

    constructor(filePath: string) {
        this.filePath = filePath;
    }
    write(message: string): void {
        //Logic to write to file
        fs.appendFileSync(this.filePath, message);
    }
}
```

Strategy UML





Taller: Programando AVAST CE

ANTIVIRUS