Template Method

Presented By:
Ahmed Elrmady

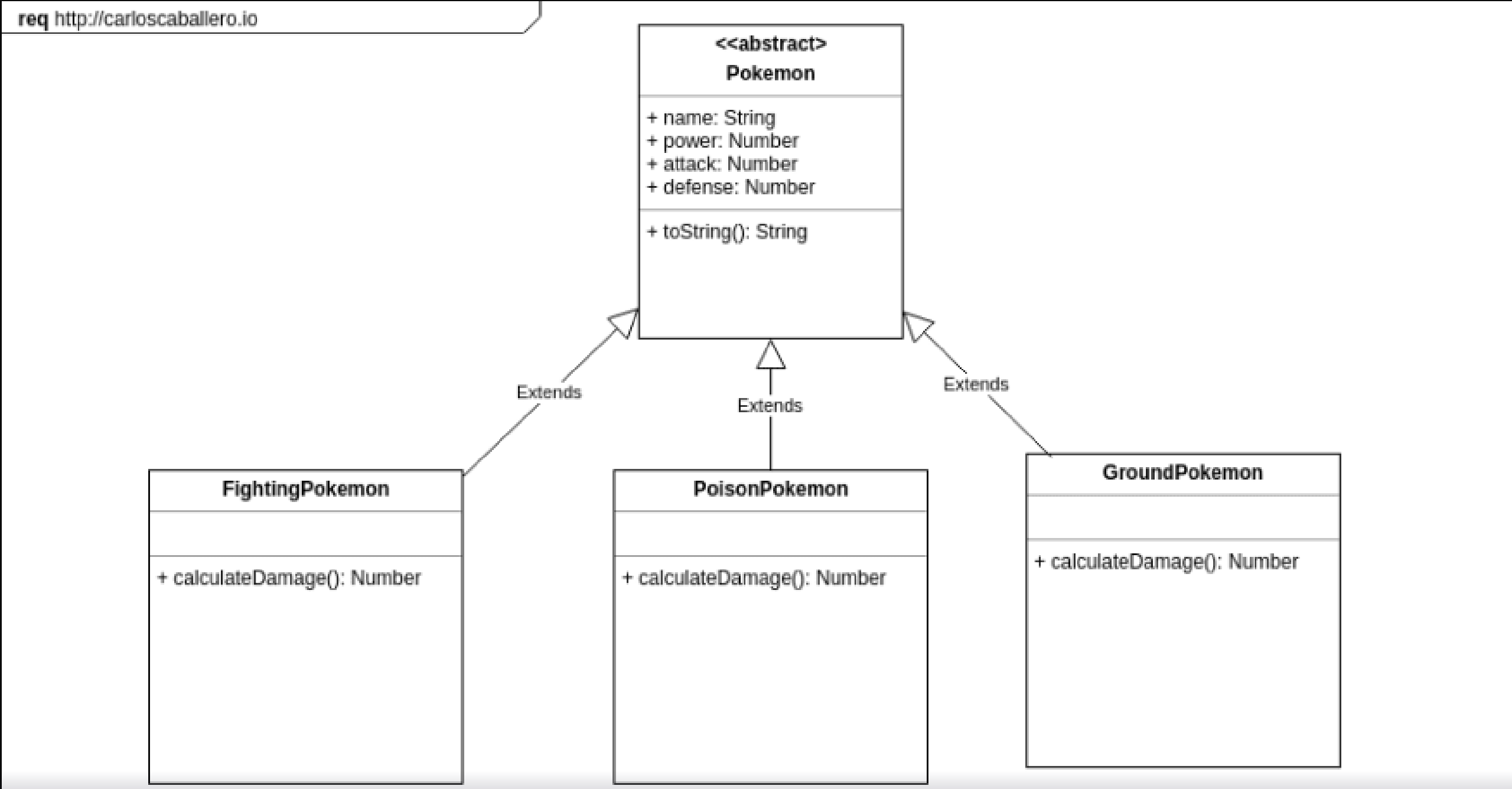
One of the **Behavioral** Design Patterns

Focus on improving the communication and assignment of responsibilities between dissimilar objects in a system

:: Defines the skeleton of an algorithm in an operation, deferring some steps to subclasses.

Why should I use it ?!

- Avoids duplicating code
- Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

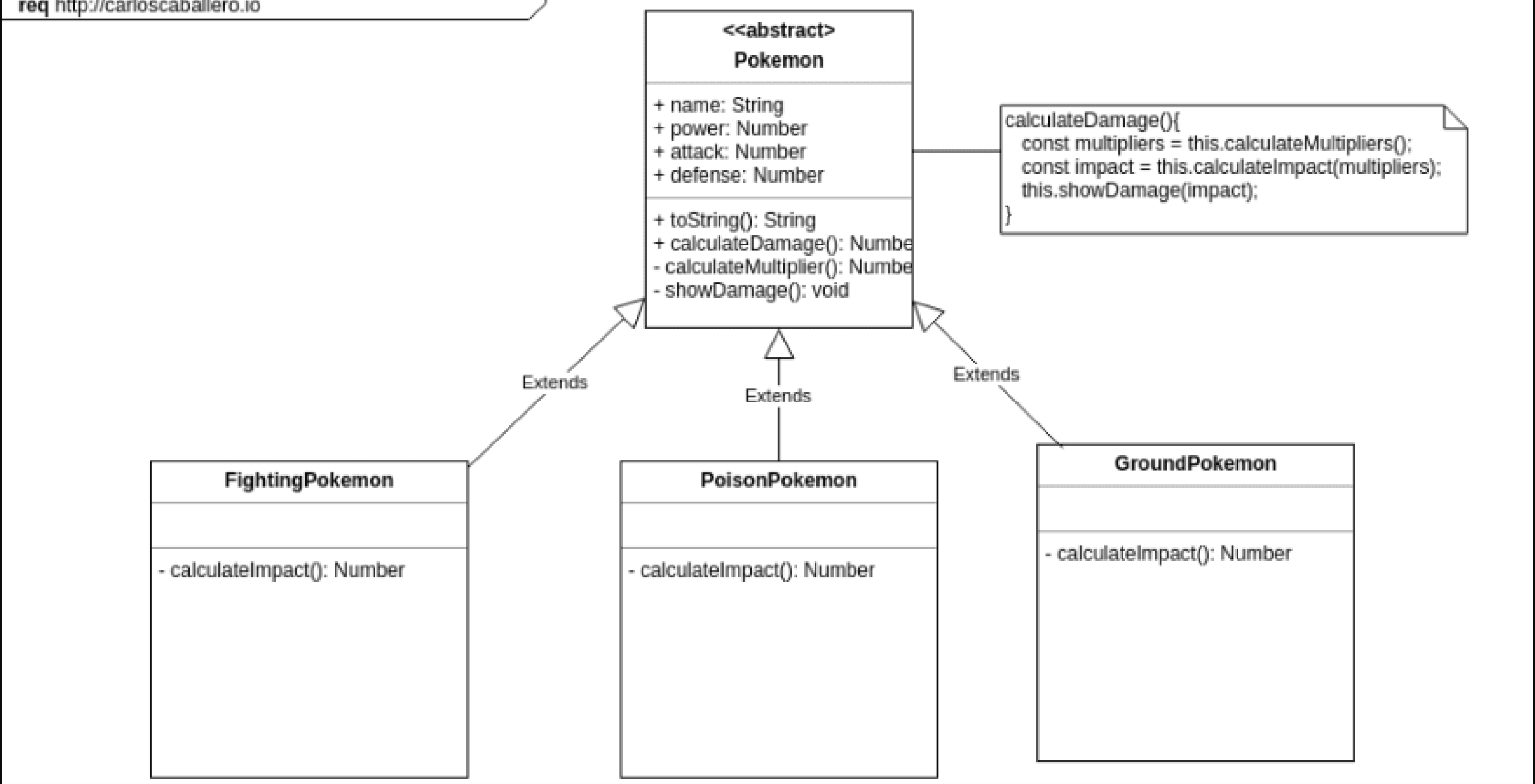

req http://carloscaballero.io
<<abstract>
Pokemon
+ name: String
+ power: Number
+ attack: Number
+ defense: Number
+ toString(): String
Extends
Extends
Extends
FightingPokemon
+ calculateDamage(): Number
PoisonPokemon
+ calculateDamage(): Number
GroundPokemon
+ calculateDamage(): Number

```
class Pokemon {
  constructor(_pokemon) {
    this.name = _pokemon.name || 'unknown';
    this.power = _pokemon.power || 1;
    this.attack = _pokemon.attack || 1;
    this.defense = _pokemon.defense || 1;
  }
  toString() {
    return `${this.name} -  power: ${this.power}; attack: ${
      this.attack
    }; defense: ${this.defense}`;
  }
}
class FightingPokemon extends Pokemon {
  constructor(_pokemon) {
    super(_pokemon);
  }
  calculateDamage() {
    const multipliers = (1 / 2) * this.power * Math.random();
    const impact = Math.floor((this.attack / this.defense) * multipliers) + 1;
    console.log('Pokemon damage is:', impact);
  }
}

class PoisonPokemon extends Pokemon {
  constructor(_pokemon) {
    super(_pokemon);
  }
  calculateDamage() {
    const multipliers = (1 / 2) * this.power * Math.random();
    const impact = Math.floor((this.attack - this.defense) * multipliers) + 1;
    console.log('Pokemon damage is:', impact);
  }
}

class GroundPokemon extends Pokemon {
  constructor(_pokemon) {
    super(_pokemon);
  }
  calculateDamage() {
    const multipliers = (1 / 2) * this.power * Math.random();
    const impact = Math.floor((this.attack + this.defense) * multipliers) + 1;
    console.log('Pokemon damage is:', impact);
  }
}
```

```
calculateDamage(){
  const multipliers = this.calculateMultipliers();
  const impact = this.calculateImpact(multipliers);
  const showDamage(impact);
}
```


req http://carloscaballero.io
<<abstract>
Pokemon
+ name: String
+ power: Number
+ attack: Number
+ defense: Number
+ toString(): String
+ calculateDamage(): Numbe
- calculateMultiplier(): Numbe
- showDamage(): void
calculateDamage(){
const multipliers = this.calculateMultipliers();
const impact = this.calculateImpact(multipliers);
this.showDamage(impact);
}
Extends
Extends
Extends
FightingPokemon
- calculateImpact(): Number
PoisonPokemon
- calculateImpact(): Number
GroundPokemon
- calculateImpact(): Number

```javascript
class Pokemon {
  constructor(_pokemon) {
    this.name = _pokemon.name || 'unknown';
    this.power = _pokemon.power || 1;
    this.attack = _pokemon.attack || 1;
    this.defense = _pokemon.defense || 1;
  }
  toString() {
    return `${this.name} -  power: ${this.power}; attack: ${
      this.attack
    }; defense: ${this.defense}`;
  }
  calculateMultiplier() {
    //Step 1 - Common
    return (1 / 2) * this.power * Math.random();
  }
  showDamage(damage) {
    // Step 3 - Common
    console.log('Pokemon damage is:', damage);
  }
  calculateDamage() {
    const multipliers = this.calculateMultiplier(); //Step 1;
    const damage = this.calculateImpact(multipliers); //Step 2;
    this.showDamage(damage); //Step 3;
  }
}
```

```javascript
class FightingPokemon extends Pokemon {
  constructor(_pokemon) {
    super(_pokemon);
  }
  calculateImpact(multipliers) {
    return Math.floor((this.attack / this.defense) * multipliers) + 1;
  }
}

class PoisonPokemon extends Pokemon {
  constructor(_pokemon) {
    super(_pokemon);
  }
  calculateImpact(multipliers) {
    return Math.floor((this.attack - this.defense) * multipliers) + 1;
  }
}

class GroundPokemon extends Pokemon {
  constructor(_pokemon) {
    super(_pokemon);
  }
  calculateImpact(multipliers) {
    return Math.floor((this.attack + this.defense) * multipliers) + 1;
  }
}
```

```
// Client-Context

const passimian = new FightingPokemon({
  name: 'Passimian',
  attack: 10,
  power: 10,
  defense: 10
});
console.log(passimian.toString());
passimian.calculateDamage();

const poipole = new PoisonPokemon({
  name: 'Poipole',
  attack: 10,
  power: 10,
  defense: 10
});
console.log(poipole.toString());
poipole.calculateDamage();

const mudsdale = new GroundPokemon({
  name: 'Mudsdale',
  attack: 10,
  power: 10,
  defense: 10
});
console.log(mudsdale.toString());
mudsdale.calculateDamage();
```