10-Strategy

Seeeeee:D夏休み勉強会
デザインパターン紹介

Katsuya Suzuki

Strategyパターンとは

- 条件によってアルゴリズム部分を切り替えたいときに有用
- よく以下のパターンと比較される
  - TemplateMethod
  - State

TemplateMethodパターンとの比較

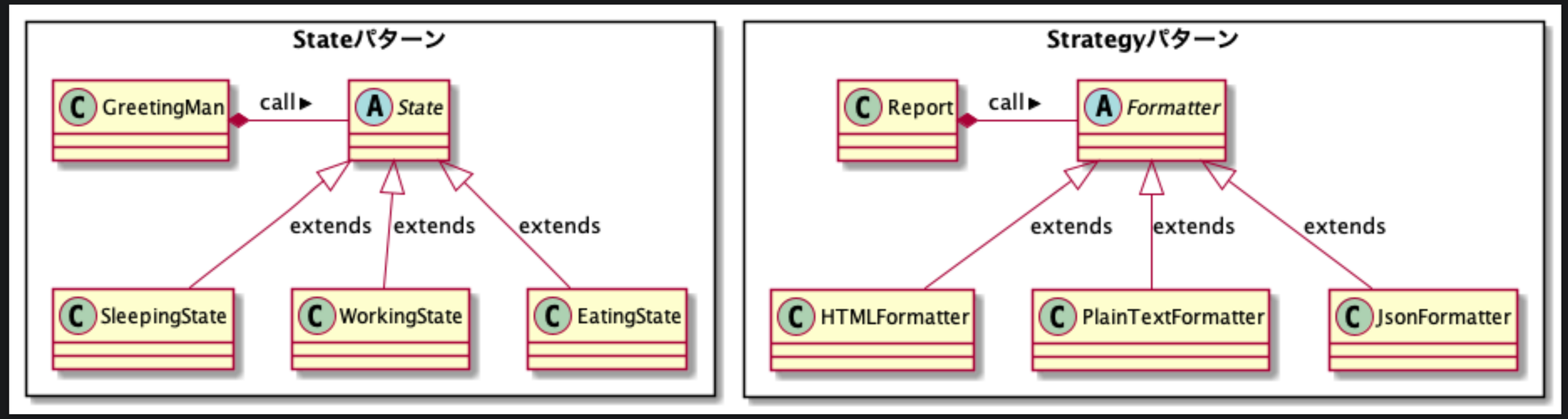
- TemplateMethod → 共通の処理を**まとめる**手法
  差分の実装はクラスの**継承**先で行う
- Strategy → ロジックの**切り替え**を工夫する手法
  差分の実装はクラスの**委譲**先で行う

TemplateMethodの，継承による副作用を解消できる

→ 親クラスにアルゴリズムが記述されているので,
別のアルゴリズムを書きたいときは別の親クラスを作る必要がある

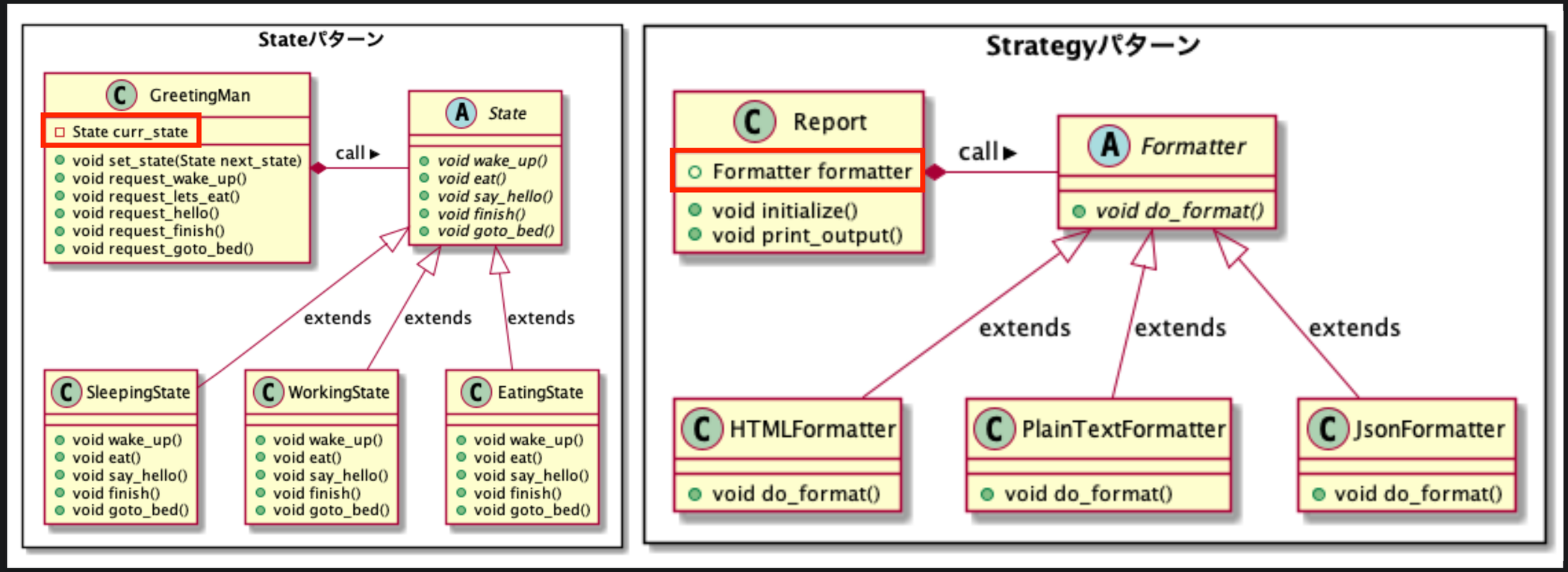
Stateパターンとの比較

- State　　→ ロジックは**内部の状態**によって切り替わる
- Strategy → ロジックは利用する側が**選択**する



クラス間の構造は同じ！

Stateパターンとの比較(詳解)



ロジックを切り替えるための変数が,

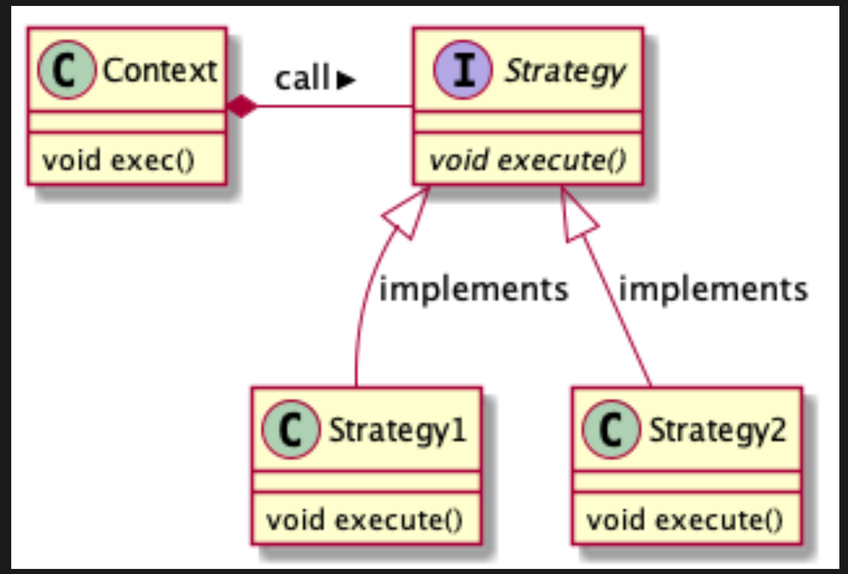
- State　　パターンはprivate　であり，勝手に遷移する　　(変数curr_state)
- Strategy　パターンはpublic　であり，ユーザ側で適宜変更可能　(変数formatter)

実装方針

- 条件によって変化する**ロジック部分を別のクラスに抽出する**
  - 1条件1クラス
- それぞれのクラスで**使用できるインタフェースは揃える**
  - クラスごとに同じメソッド名にする
  - JavaとC#はこれをinterfaceを用いて実装する

ざっくり
こんな感じ

ってかこれが全て




```
interface Strategy {
    void execute();  // このメソッド名で呼ばれるから統一してね
}

class Strategy1 implements Strategy {
    void execute() {
        // アルゴリズム1パターン目を実装
    }
}

class Strategy2 implements Strategy {
    void execute() {
        // アルゴリズム2パターン目を実装
    }
}

class Context {
    void exec(Strategy s) {
        s.execute();      // 実行時までどのexecute()を呼ぶのか未定
    }
}

Context context = new Context();  // 渡すインスタンスによって
context.exec(new Strategy1());    // アルゴリズムが切り替わる！
```

プログラム例

- 報告書を作成するプログラムを作る
  - TemplateMethodと比較するために同じ題材で
- 書く内容は共通している
- 複数の形式で提出する必要がある
  - .html
  - .txt
  - .json　　(←new!!)
- VSCodeで実際のコードを確認します！

提出形式が増えたら

- Formatterクラスを継承して，新しい提出形式に沿った記述アルゴリズムを実装する
- 新しく実装したクラスのインスタンスをReport.formatterに代入すれば切り替わる

プログラム例2

- 画像を表示するプログラムを作る
- 表示方法を3パターン実装したい
  - カラー
  - グレー
  - モザイク
- VSCodeへ

アルゴリズムの選択を増やすことが容易に

例えば

- 顔認識した結果を表示したい
- ノイズを除去したい
- 明るさやコントラストを変えたい

入出力は他のクラスの担当なので，アルゴリズムだけ実装するような
クラスを増やせば良い！

まとめ

- ロジックの切り替えが容易になる
- if-else増えたなって思ったら検討してみては