22 Command パターン

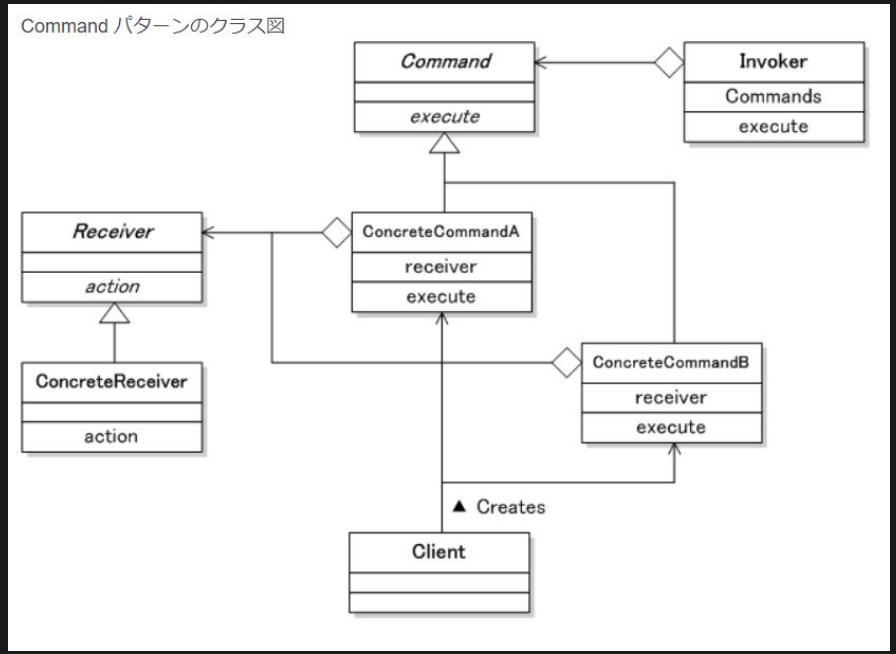
Seeeeee:D デザインパターン勉強会

Commandパターンとは

- 関数呼び出しのためのパラメータ(ユーザー名・時間など)をCommandオブジェクトに一時保存できる。
- コマンドをオブジェクトとして扱うのでコマンドのデータ構造を作れる。(木構造・スタック・キューなど)
- データ構造により、履歴やUndoを実現できる。

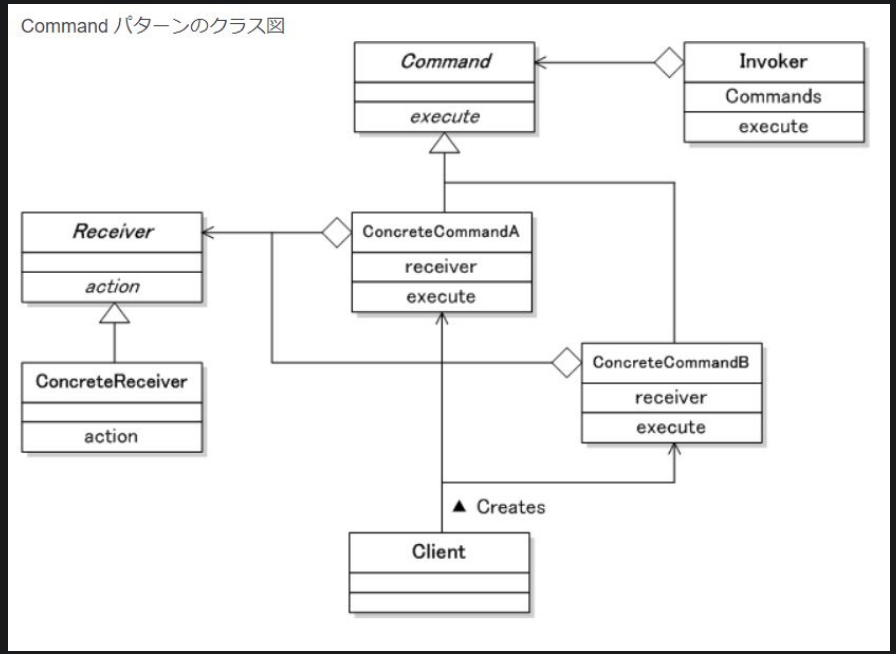
登場人物1

- Command
  - 命令のインターフェイスを定義
- Invoker
  - Commandを呼び出す。
- ConcreteCommand
  - 命令の具体的な実装



登場人物2

- Receiver
  - 処理対象のインターフェイス
- Client
  - 利用者(今回はMain)



Command.ts

```ts
import Receiver from "./Receiver";

export default interface Command {
  setReceiver(receiver: Receiver): void;
  execute(): void;
}
```

- Command型を定義
  - setReceiverは名前の通り
  - executeはコマンドの実行

Inboker.ts

- Commandインターフェイスを呼び出す
- ConcreteCommandを保持し、履歴やUndo機能を提供。

```
export default class Invoker {
  private commands: Command[] = [];

  addCommand(command: Command): void {
    this.commands.push(command);
  }
  undoCommand(): void {
    console.log(this.commands.pop())
  }
  execute(): void {
    for (const command of this.commands) {
      command.execute();
    }
  }
}
```

ConcreteCommand.ts

- Commandの実装
- receiverを変えることで実行対象を変更できる
- executeはコマンドの実行

```
export default class ConcreteCommand
implements Command {
  private number: number;
  private receiver: Receiver;

  constructor(number: number) {
    this.number = number;
  }

  setReceiver(receriver: Receiver): void {
    this.receiver = receriver;
  }

  execute(): void {
    this.receiver.action(this.number *
this.number);
  }
}
```

ConcreteCommand2.ts

- executeを少し変えた。
- こんな感じで複数種のコマンドを定義できる。

```
export default class ConcreteCommand implements Command {
  private number: number;
  private receiver: Receiver;

  constructor(number: number) {
    this.number = number;
  }
  setReceiver(receriver: Receiver): void {
    this.receiver = receriver;
  }
  execute(): void {
      this.receiver.action(this.number+this.number);
  }
}
```

Receiver.ts

```
export default class Receiver {
    action(number: number): void {
    console.log(number);
    }
  }
```

- 処理対象オブジェクト。
  - 出力やDB操作など?
- 複数作る事もできる。

Main.ts(Client)

- ConcreteCommandの初期設定
- Invokerの操作を行う

```
const receiver: Receiver = new Receiver;
const invoker: Invoker = new Invoker;

//コマンドインスタンスを作成。
const threeCommand: Command = new ConcreteCommand(3);
//レシーバー(命令の受取り手(インターフェイス/API)をセット)
threeCommand.setReceiver(receiver);
const fiveCommand: Command = new ConcreteCommand(5);
fiveCommand.setReceiver(receiver);

const tenSumCommand: Command = new ConcreteCommand2(10);
tenSumCommand.setReceiver(receiver);

//コマンドをキューに追加
invoker.addCommand(fiveCommand);

//コマンドをキューから実行
invoker.execute();
invoker.undoCommand()

invoker.printHistory()
invoker.execute();
```

感想

- なかなか個人で使うことは難しいかも(大規模アプリケーション向き??)
- CG基礎やプログラミング実習のOpenGLで似たことやった気がする。
- wordとかペイントは文字や座標を引数にConcreteCommandインスタンスつくってるのかな？