Command-Pattern

Carsten Gips (HSBI)

Motivation

```
public class InputHandler {
    public void handleInput() {
        switch (keyPressed()) {
            case BUTTON_W -> hero.jump();
            case BUTTON_A -> hero.moveX();
            case ...
            default -> { ... }
        }
    }
}
```

Problem: Starre Zuordnung

Auflösen der starren Zuordnung über Zwischenobjekte

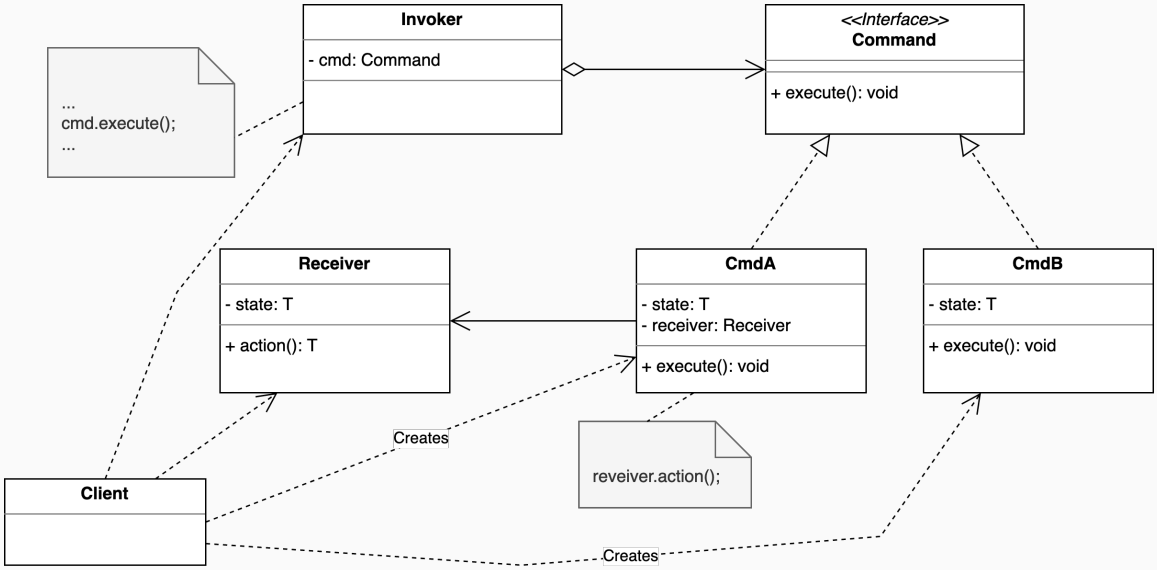
```
public interface Command { void execute(); }

public class Jump implements Command {
    private Entity e;
    public void execute() { e.jump(); }
}

public class InputHandler {
    private final Command wbutton = new Jump(hero);  // Über Ctor/Methoden setzen!
    private final Command abutton = new Move(hero);  // Über Ctor/Methoden setzen!

    public void handleInput() {
        switch (keyPressed()) {
            case BUTTON_W -> wbutton.execute();
            case BUTTON_A -> abutton.execute();
            case ...
            default -> { ... }
        }
    }
}
```

Command: Objektorientierte Antwort auf Callback-Funktionen



Undo

```
public class Move implements Command {
    private Entity e;
    private int x, y, oldX, oldY;

    public void execute() { oldX = e.getX();  oldY = e.getY();  x = oldX + 42;  y = oldY;  e.moveTo(x, y); }
    public void undo() { e.moveTo(oldX, oldY); }
    public Command newCommand(Entity e) { return new Move(e); }
}

public class InputHandler {
    private final Command wbutton;
    private final Command abutton;
    private final Stack<Command> s = new Stack<>();

    public void handleInput() {
        Entity e = getSelectedEntity();
        switch (keyPressed()) {
            case BUTTON_W -> { s.push(wbutton.newCommand(e)); s.peek().execute(); }
            case BUTTON_A -> { s.push(abutton.newCommand(e)); s.peek().execute(); }
            case BUTTON_U -> s.pop().undo();
            case ...
            default -> { ... }
        }
    }
}
```

Wrap-Up

Command-Pattern: Kapsele Befehle in ein Objekt

- `Command`-Objekte haben eine Methode `execute()` und führen darin Aktion auf Receiver aus
- `Receiver` sind Objekte, auf denen Aktionen ausgeführt werden (Hero, Monster, ...)
- `Invoker` hat `Command`-Objekte und ruft darauf `execute()` auf
- `Client` kennt alle und baut alles zusammen

Objektorientierte Antwort auf Callback-Funktionen

LICENSE