

Internetové aplikace

Projekt 1 – Objekty v jazyce JavaScript

Command

Autor: Jakub Sadílek

Login: xsadil07

Datum: 7.3.2021

Zadání

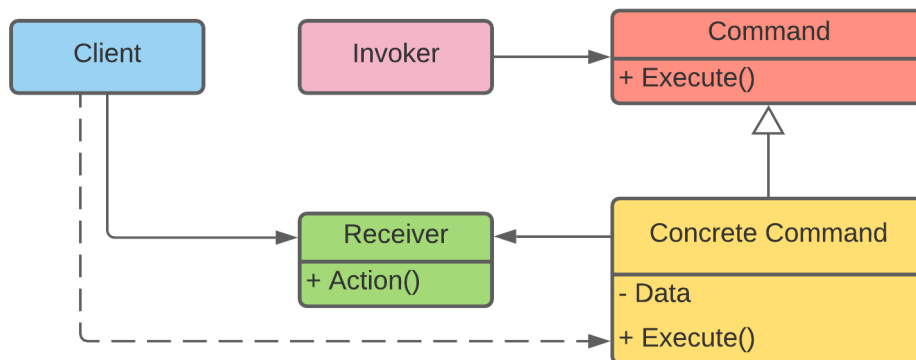
Cílem tohoto projektu je seznámit se a pochopit objekty v jazyce JavaScript a porozumět návrhovým vzorům. K zadanému návrhovému vzoru si nastudovat informace a vybrat si libovolnou vhodnou doménu, ve které lze návrhový vzor uplatnit a následně implementovat.

Návrhový vzor Command

Návrhový vzor Command patří do kategorie vzorů chování. Jedná se o zapouzdřený požadavek na samotný objekt, který obsahuje všechny podstatné informace k jeho provedení. To nám umožňuje provádět abstraktní operace nad požadavky, uchovávat jejich historii či vracet efekt jednotlivých operací.

Entity vyskytující se tomto vzoru jsou:

- **Client** – Vytváří objekty typu Concrete Command a specifikuje jeho příjemce.
- **Command** – Deklaruje rozhraní pro vykonání požadavku.
- **Concrete Command** – Obsahuje parametry a operace pro vykonání konkrétního požadavku. Implementuje metodu `execute()` pro jeho vykonání na příjemci.
- **Receiver** – Přijímá commandy a je zodpovědný za jejich zpracování.
- **Invoker** – Vykoná požadavek pomocí volání metody `Execute()` objektu Concrete Command.



Obr. 1: Schéma návrhového vzoru Command

Aplikace

Jako doménu pro implementaci návrhového vzoru Command jsem zvolil kalkulačku, jelikož zde lze přehledně znázornit implementaci vzoru, kde jednotlivé operace kalkulačky jsou zapouzdřeny do Commandů. Zároveň je zde možné implementovat funkce nad příkazy jako jsou `undo()` či `redo()` a má tedy smysl si jednotlivé commandy ukládat. Pro rozhraní kalkulačky jsem se rozhodl inspirovat prací v assembleru. Uživatel postupně zadává příkazy (instrukce), kde každý příkaz ihned po odeslání je zpracován a uživateli je zobrazen výsledek.

Implementované operace jsou:

- Help – Vypíše nápovědu aplikace.
- Quit – Ukončí aplikaci.
- Undo – Vráť efekt poslední operace.
- Redo – Aplikuje efekt poslední vrácené operace.
- Neg – Změní znaménko výsledku.
- Reset – Vynuluje výsledek.
- Set *<value>* – Nastaví výsledek na zadanou hodnotu.
- Add *<value>* – Přičte k výsledku hodnotu.
- Sub *<value>* – Odečte od výsledku hodnotu.
- Mul *<value>* – Vynásobí výsledek zadanou hodnotou.
- Div *<value>* – Vydělí výsledek zadanou hodnotou.
- Sim *<value>* – Vygeneruje posloupnost příkazů a zobrazí výsledek uživateli. Náhodná simulace pro demonstraci použití aplikace.

Implementace

Projekt je implementován v modulech `library.mjs` a `model.mjs`. Modul `library.mjs` obsahuje obecné API k využití kalkulačky jako je například abstraktní třída `Command`, která definuje základní rozhraní pro jednotlivé operace kalkulačky. Všechny implementované operace se zde nachází také a jsou rozšířeny o metodu `undo()`, která vrátí stav kalkulačky do stavu před provedením dané operace. Další abstraktní třída, která se zde nachází je `Invoker`, který spouští jednotlivé požadavky a implementuje ho třída `Calculator`, která zároveň udržuje podstatná data ke správnému fungování kalkulačky (výsledek, historii a seznam vrácených operací) a rozšiřuje funkčnost `Invokeru` o operace `undo()` a `redo()`. Lze si povšimnout, že třída `Receiveru` se tady nenachází, jelikož v tomto případě (obecně v JavaScriptu) je výhodné takto spíše využít `Invoker`, jelikož zde všechny funkce a objekty jsou víceméně commandy samotné.

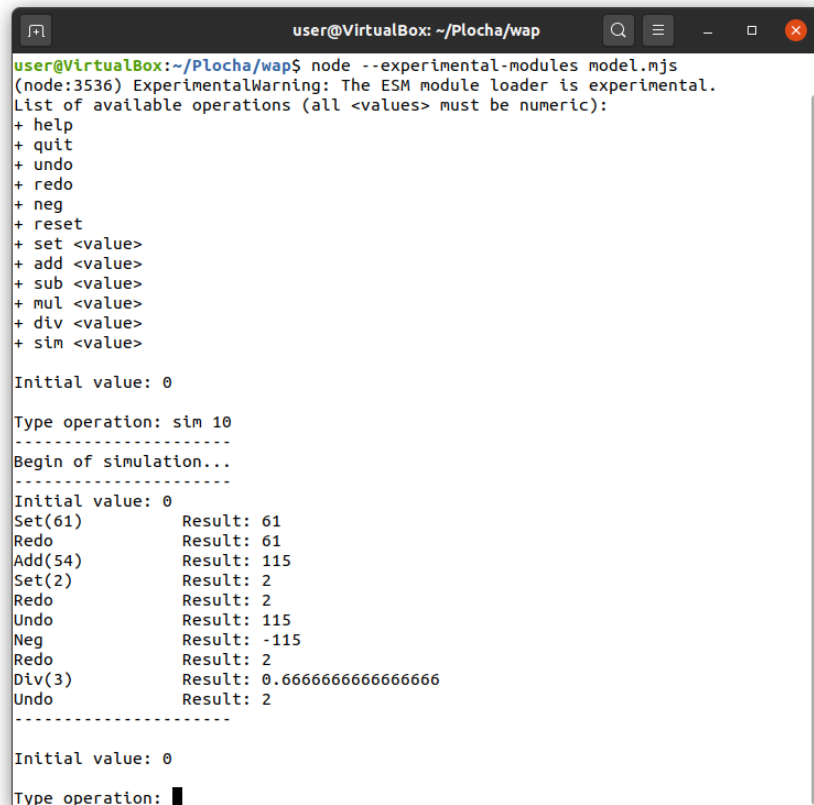
Modul `model.mjs` využívá obecnou API kalkulačky a implementuje rozhraní, kterým aplikace disponuje. Obsah tohoto modulu slouží jako demonstrační využití API, jinak lze použít i jiné rozhraní např. s GUI. Konkrétně je zde udržována instance kalkulačky (`Client`) a po zpracování vstupu uživatele jsou invokovány konkrétní commandy. Také se zde nachází funkce simulace pro jednodušší představení funkčnosti aplikace.

Spuštění

Projekt byl implementován v prostředí NodeJS ve verzi v10.19.0. Po nainstalování tohoto prostředí lze projekt spustit modulem `model.mjs` následujícím příkazem:

```
$> node --experimental-modules model.mjs
```

Pro snadnou demonstraci použití je možné použít vestavěnou funkci „sim“ viz obrázek 2.



```
user@VirtualBox: ~/Plocha/wap
user@VirtualBox:~/Plocha/wap$ node --experimental-modules model.mjs
(node:3536) ExperimentalWarning: The ESM module loader is experimental.
List of available operations (all <values> must be numeric):
+ help
+ quit
+ undo
+ redo
+ neg
+ reset
+ set <value>
+ add <value>
+ sub <value>
+ mul <value>
+ div <value>
+ sim <value>

Initial value: 0

Type operation: sim 10
-----
Begin of simulation...
-----
Initial value: 0
Set(61)      Result: 61
Redo         Result: 61
Add(54)      Result: 115
Set(2)       Result: 2
Redo         Result: 2
Undo         Result: 115
Neg          Result: -115
Redo         Result: 2
Div(3)       Result: 0.6666666666666666
Undo         Result: 2
-----

Initial value: 0

Type operation: █
```

Obr. 2: Příklad spuštění aplikace

Závěr

Zadání projektu bylo podle mě velmi otevřené a bylo zde spousta místa pro vlastní nápady a inovace. Jeho náročnost byla přiměřená a projekt mě bavil. Osobně bych ale kvůli využití návrhového vzoru Command preferoval implementaci v nějakém staticky typovaném jazyce (např. C++).

Zdroje

1. https://en.wikipedia.org/wiki/Command_pattern
2. <https://www.oreilly.com/library/view/learning-javascript-design/9781449334840/ch09s08.html>
3. <https://jsmanifest.com/command-design-pattern-in-javascript/>
4. <https://nodejs.org/api/>