Как и когда использовать поведенческий шаблон проектирования «Цепочка обязанностей»

Содержание

[Задание 1. Первая реализация 3](#_Toc94282190)

[Задание 2. Вторая реализация – начинаем улучшать код 6](#_Toc94282191)

[Задание 3. Третья реализация – избавляемся от дублирующегося кода 7](#_Toc94282192)

[Задание 4. Четвертая реализация – почти на финишной прямой 8](#_Toc94282193)

[Задание 5. Построение динамической цепочки 9](#_Toc94282194)

# Задание 1. Первая реализация

1. Создайте интерфейс **Handler** в файле **src/handler.interface.ts**
2. Интерфейс должен содержать два метода:
   1. setNext(handler: Handler): Handler
   2. handle(fileName: string): void;
3. Экспортируйте интерфейс как тип
4. Создайте папку **src/step-1/handlers,** а в ней создайте файлы для всех обработчиков, которые необходимо реализовать как классы:
   1. docx-file-handler.class.ts
   2. jpg-file-handler.class.ts
   3. mp3-file-handler.class.ts
   4. mpg4-file-handler.class.ts
   5. txt-file-handler.class.ts
   6. xlsx-file-handler.class.ts
   7. zip-file-handler.class.ts
5. Назовите классы обработчиков следующими именами:
   1. DocxHandler
   2. JpgHandler
   3. Mp3Handler
   4. Mpg4Handler
   5. TxtHandler
   6. XlsxHandler
   7. ZipHandler
6. Каждый класс должен:
   1. реализовывать интерфейс **Handler**, то есть содержать методы **setNext и handle**
   2. содержать **private nextHandler!: Handler;**
7. Для реализации метода **setNext** используйте следующий фрагмент кода

setNext(handler: Handler): Handler {

this.nextHandler = handler;

return handler;

}

1. Для реализации метода **handle** используйте следующий фрагмент кода

handle(fileName: string): void {

console.log(`Hi, I am ${Object.getPrototypeOf(this).constructor.name}`);

// измените условие

if (fileName.endsWith('.zip')) {

// измените текст

console.log('I am running WinZip... I pass file to it.');

} else if (this.nextHandler) {

console.log(`I can not process ${fileName}. I pass it to the next ${Object.getPrototypeOf(this.nextHandler).constructor.name}`);

this.nextHandler.handle(fileName);

} else {

console.log(`No Handler for ${fileName}`);

}

}

1. Во всех классах внесите изменение в условие в первой ветке оператора if, а также замените сообщение соответствующим типу файла из первоначальной конструкции **if – else if – else if … else**
2. Создайте файл **src/step-1/handlers/index.ts** и добавьте в него реэкспорт всех модулей с классами обработчиков. Используйте следующий фрагмент кода

export \* from './docx-file-handler.class';

export \* from './jpg-file-handler.class';

export \* from './mp3-file-handler.class';

export \* from './mpg4-file-handler.class';

export \* from './txt-file-handler.class';

export \* from './xlsx-file-handler.class';

export \* from './zip-file-handler.class';

1. Создайте файл **src/client.ts**, а в нем функцию **runHandlersChain.** Используйте следующий фрагмент кода:

import { Handler } from './handler.interface';

import \* as Handlers from './step-1/handlers';

export function runHandlersChain() {

// файлы для обработки

const files = ['file1.docx', 'file2.zip', 'file3.mp3', 'file4.avi'];

// объект содержит всевозможные обработчики

const handlers = {

docx: new Handlers.DocxHandler(),

jpg: new Handlers.JpgHandler(),

mp3: new Handlers.Mp3Handler(),

mpg4: new Handlers.Mpg4Handler(),

txt: new Handlers.TxtHandler(),

xlsx: new Handlers.XlsxHandler(),

zip: new Handlers.ZipHandler()

}

// создать цепочку из всех обработчиков

handlers.docx.setNext(handlers.jpg)

.setNext(handlers.mp3)

.setNext(handlers.mpg4)

.setNext(handlers.txt)

.setNext(handlers.xlsx)

.setNext(handlers.zip);

// запустить обработку файлов

console.log('---Статическая цепочка---');

files.forEach(file => handlers.docx.handle(file));

}

1. Импортируйте данную функцию в приложение и запустите ее. Проанализируйте вывод в консоль.

# Задание 2. Вторая реализация – начинаем улучшать код

1. Создайте папку **src/step-2/handlers** и скопируйте в нее содержимое папки **src/step-1/handlers**.
2. Создайте файл **src/step-2/abstract-handler.class.ts** и добавьте в него следующий фрагмент кода:

import type { Handler } from './../handler.interface';

export abstract class AbstractHandler {

protected nextHandler!: Handler;

}

1. Внесите изменения во все классы обработчиков:
   1. все классы должны расширять базовый класс **AbstractHandler** и реализовывать интерфейс **Handler**
   2. удалите из классов приватное свойство **nextHandler.** Теперь вместо него будет использоваться свойство из базового класса.
2. Внесите изменения в файл **src/client.ts**: Вместо

import \* as Handlers from './step-1/handlers';

используйте

import \* as Handlers from './step-2/handlers';

Проанализируйте вывод в консоль.

# Задание 3. Третья реализация – избавляемся от дублирующегося кода

1. Создайте папку **src/step-3** и скопируйте в нее содержимое папки **src/step-2**.
2. Внесите изменения в класс **AbstractHandler** – добавьте метод setNext. Используйте следующий фрагмент кода:

setNext(handler: Handler): Handler {

this.nextHandler = handler;

return handler;

}

1. Удалите метод **setNext** во всех классах обработчиках.
2. Внесите изменения в файл **src/client.ts**: Вместо

import \* as Handlers from './step-2/handlers';

используйте

import \* as Handlers from './step-3/handlers';

Проанализируйте вывод в консоль.

# Задание 4. Четвертая реализация – почти на финишной прямой

1. Создайте папку **src/step-4** и скопируйте в нее содержимое папки **src/step-3**.
2. Внесите изменения в класс **AbstractHandler.** Теперь этот класс должен реализовывать интерфейс **Handler.**
3. Добавьте метод **handle**, используйте следующий фрагмент кода:

handle(fileName: string): void | null {

if (this.nextHandler) {

console.log(`I can not process ${fileName}. I pass it to the next ${Object.getPrototypeOf(this.nextHandler).constructor.name}`);

return this.nextHandler.handle(fileName);

}

console.log(`No Handler for ${fileName}`);

return null;

}

1. Измените модификатор доступа для свойства **nextHandler** с **protected** на **private**
2. Внесите изменения в метод handle во всех классах обработчиков, используйте следующий фрагмент кода:

handle(fileName: string): void {

console.log(`Hi, I am ${Object.getPrototypeOf(this).constructor.name}`);

// измените условие и текст вывода

if (fileName.endsWith('.docx')) {

console.log('I am running MS Word... I pass file to it.');

return;

}

super.handle(fileName);

}

1. Удалите интерфейс **Handler** из списка implements всех классов обработчиков.
2. Внесите изменения в файл **src/client.ts**: Вместо

import \* as Handlers from './step-3/handlers';

используйте

import \* as Handlers from './step-4/handlers';

1. Проанализируйте вывод в консоль.

# Задание 5. Построение динамической цепочки

1. Добавьте в функцию **runHandlersChain** следующий фрагмент кода:

// создать динамическую цепочку обработчиков на основании расширения файлов

let handler: Handler; // ссылка на первый обрабботчик в цепочке для старта

let currentHandler: Handler; // ссылка на текущий обрабботчик для построения цепочки обработчиков

files.forEach((fileName, index) => {

const [, ext] = fileName.split('.');

if (index === 0) {

handler = handlers[ext];

currentHandler = handlers[ext];

} else {

currentHandler = currentHandler.setNext(handlers[ext]);

}

});

// запустить обработку файлов

console.log('---Динамическая цепочка---');

files.forEach(file => handler.handle(file));

1. Проанализируйте вывод в консоль.