

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий
институт
Кафедра «Информатика»
кафедра

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3

Методология тестирования
тема

Вариант 4

Преподаватель

А. С. Кузнецов

подпись, дата

Студент

А. М. Сотниченко

подпись, дата

Красноярск 2021

1 Цель работы

На конкретных примерах ознакомиться с базовыми методами блочного тестирования программного обеспечения.

2 Общая постановка задачи

Продemonстрировать понимание и применение на практике ключевых понятий, рассмотренных в этой работе.

3 Запуск проекта

Чтобы запустить проект с начала надо установить NodeJS и npm, далее перейти в директорию с проектом и выполнить две команды:

- `npm install;`
- `npm run test.`

4 Ход работы

Создадим Класс Parser и напомним метод `processLine`.

```

export class Parser {
  static processLine(line: string): Token {
    line.replace("\\s+", "")

    switch (line) {
      case "+":
        return Token.Plus
      case "-":
        return Token.Minus
      case "*":
        return Token.Mult
      case "/":
        return Token.Div
      case "":
        return Token.EmptyString
    }

    const val = Rational.parse(line)
    if (val) {
      return Token.Number
    }

    return Token.Error
  }
}

```

Рисунок 1 – Класс Parser

Включим отображение покрытия кода тестами, добавим в конфиг Jest строчку “collectCoverage: true”. Теперь начнем покрывать код тестами.

```

describe("Parse.processLine", () => {
  test("Plus", () => {
    expect(Parser.processLine("+")).toBe(Token.Plus)
  })
})

```

Рисунок 2 – Частичное покрытие метода processLine

PASS src/parser.test.ts					
Parse.processLine					
✓ Plus (4 ms)					
File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	35	23.07	15.38	35	
parser.ts	61.9	33.33	100	61.9	21-35
rational.ts	5.26	0	0	5.26	14-108
Test Suites: 1 passed, 1 total					
Tests: 1 passed, 1 total					
Snapshots: 0 total					
Time: 4.537 s, estimated 7 s					
Ran all test suites.					

Рисунок 3 – Результат тестирования

Покроем класс Parser тестами до конца, также скопируем тесты для класса Rational из прошлой работы.

```

describe("Parse.processLine", () => {
  test("Plus", () => {
    expect(Parser.processLine("+")).toBe(Token.Plus)
  })

  test("Minus", () => {
    expect(Parser.processLine("-")).toBe(Token.Minus)
  })

  test("Mult", () => {
    expect(Parser.processLine("*")).toBe(Token.Mult)
  })

  test("Div", () => {
    expect(Parser.processLine("/")).toBe(Token.Div)
  })

  test("EmptyString", () => {
    expect(Parser.processLine("")).toBe(Token.EmptyString)
  })

  test("Number", () => {
    expect(Parser.processLine("1/3")).toBe(Token.Number)
  })

  test("Error", () => {
    expect(Parser.processLine("error")).toBe(Token.Error)
  })
})

```

Рисунок 4 – Покрытие метода processLine

PASS src/rational.test.ts (7.359 s)
PASS src/parser.test.ts (8.893 s)

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
parser.ts	100	100	100	100	
rational.ts	100	100	100	100	

Test Suites: 2 passed, 2 total
 Tests: 19 passed, 19 total
 Snapshots: 0 total
 Time: 10.546 s
 Ran all test suites.

Рисунок 5 – Результат тестирования

Добавим метод `parse` и напомним тесты.

```
static parse(line: string): Rational | null {
  const result = line.match(/^((?:[+-]?)(\d+(?:\.\d*)?|\.\d+)(?:\s*/\s*(?:[+-]?)(\d+(?:\.\d*)?|\.\d+))?)$/i)

  if (result) {
    const [, dividendSign, dividendNum, divisorSign, divisorNum] = result
    const dividend = parseFloat(dividendSign + dividendNum)
    const divisor = parseFloat(divisorSign + divisorNum)
    return new Rational(dividend, divisor)
  }

  return null
}
```

Рисунок 6 – Метод `parse`

```
describe("Rational.parse", () => {
  test("1/11", () => {
    const a = Rational.parse("1/11")
    const b = new Rational(1, 11)
    expect(a?.equals(b)).toBeTruthy()
  })
  test("-2/-3", () => {
    const a = Rational.parse("-2/-3")
    const b = new Rational(-2, -3)
    expect(a?.equals(b)).toBeTruthy()
  })
  test("1.1/2.2", () => {
    const a = Rational.parse("1/11")
    const b = new Rational(1, 11)
    expect(a?.equals(b)).toBeTruthy()
  })
  test("1/", () => {
    const a = Rational.parse("1/")
    expect(a).toBe(null)
  })
  test("empty string", () => {
    const a = Rational.parse("")
    expect(a).toBe(null)
  })
})
```

Рисунок 7 – Тесты метода `parse`

Добавим метод `tripleAnd` и напишем тесты. Воспользуемся параметризированным тестированием.

```
static tripleAnd(a: boolean, b: boolean, c: boolean): boolean {  
  return a && b && c  
}
```

Рисунок 8 – Метод `tripleAnd`

```
describe("Parse.tripleAnd", () => {  
  const params = [  
    [false, false, false, false],  
    [false, false, true, false],  
    [false, true, false, false],  
    [false, true, true, false],  
    [true, false, false, false],  
    [true, false, true, false],  
    [true, true, false, false],  
    [true, true, true, true],  
  ]  
  each(params).test("tripleAnd variant",  
    (a: boolean, b: boolean, c: boolean, expected: boolean) => {  
      expect(Parser.tripleAnd(a, b, c)).toBe(expected)  
    })  
})
```

Рисунок 9 – Тесты метода `tripleAnd`

В конце я заметит, что Jest также генерирует web-страницу с результатами тестирования, рассмотрим поподробнее.

All files

100% Statements 48/48 100% Branches 13/13 100% Functions 13/13 100% Lines 48/48

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements	Branches	Functions	Lines
parser.ts	<div></div>	100%	21/21	100%	2/2
rational.ts	<div></div>	100%	19/19	100%	11/11

Рисунок 10 – Страница с общими результатами тестирования

All files parser.ts

100% Statements 21/21

100% Branches 9/9

100% Functions 2/2

100% Lines 21/21

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

```
1 1x import { Rational } from "./rational"
2
3 1x export enum Token {
4 1x   Number,
5 1x   Plus,
6 1x   Minus,
7 1x   Mult,
8 1x   Div,
9 1x   EmptyString,
10 1x  Error,
11 }
12
13 1x export class Parser {
14   static processLine(line: string): Token {
15     7x   line.replace("\\s+", "")
16
17     7x   switch (line) {
18     case "+":
19       1x   return Token.Plus
20     case "-":
21       1x   return Token.Minus
22     case "*":
23       1x   return Token.Mult
24     case "/":
25       1x   return Token.Div
26     case "":
27       1x   return Token.EmptyString
28   }
29
30   2x   const val = Rational.parse(line)
31   2x   if (val) {
32     1x   return Token.Number
33   }
34
35   1x   return Token.Error
36 }
37 }
38
```

Рисунок 11 – Страница с результатами тестирования parser.ts

5 Вывод

В данной работе мы ознакомились с методологией тестирования и оценкой результатов тестирования на языке TypeScript(JavaScript) с применением фреймворка Jest.