

Visitor

دیزاین پترن هادر React

Behavioral Patterns

به زبان ساده همراه با مثال های عملی
کاربرد دیزاین پترن ها برای برنامه نویسان React



Behavioral Patterns

Visitor

”

الگوی **Visitor** به ما اجازه می دهد عملیات مختلفی را روی مجموعه ای از اشیاء بدون تغییر در کلاس آن ها اعمال کنیم. از این الگو می توانیم برای پردازش داده های ساختاریافته یا مدیریت کامپوننت های پیچیده استفاده کنیم.

فرض کنید می خواهیم یک برنامه کوچک برای تناسب اندازی بسازیم که بتواند BMI و BMR و TDEE را بر اساس پارامترهای ورودی محاسبه نماید.

این کار را به گونه ای پیاده سازی می کنیم که نیاز به تغییر کلاس های داده اصلی نباشد و منطق از کلاس های داده کاملاً جدا باشند.

آشنایی با مفاهیم TDEE و BMR و BMI

در این بخش ابتدا با این مفاهیم که در رابطه با حوزه تناسب اندام می باشند آشنا می شویم.

BMI

شاخص توده بدنی

ارزیابی وزن بدن نسبت به قد

$$BMI = \frac{Weight}{Height^2} \rightarrow$$

- کمتر از 18.5: کم وزن
- 18.5 تا 24.9: وزن نرمال
- 25 تا 29.9: اضافه وزن
- 30 و بیشتر: چاقی

BMR

متابولیسم پایه

حداقل کالری لازم که بدن برای عملکردهای پایه (تنفس، گردش خون و حفظ دمای بدن) در حالت استراحت نیاز دارد.

$$BMR = 88.36 + (13.4 \times Weight) + (4.8 \times Height) - (5.7 \times Age) \rightarrow$$

محاسبه BMR در مردان

$$BMR = 447.6 + (9.2 \times Weight) + (3.1 \times Height) - (4.3 \times Age) \rightarrow$$

محاسبه BMR در زنان

TDEE

کل انرژی مصرفی روزانه

میزان کالری است که بدن در یک روز، با در نظر گرفتن سطح فعالیت های فیزیکی، مصرف می کند.

$$TDEE = BMR \times ActivityLevel$$

- 1.2 (کم تحرک): Sedentary
- 1.375 (فعالیت سبک): Light
- 1.55 (فعالیت متوسط): Moderate
- 1.725 (فعالیت زیاد): Active
- 1.9 (خیلی فعال): Very Active

Activity
Levels

بخش

دو Enum برای ActivityLevel و Gender به شکل زیر می سازیم.

Enums

```
export enum Gender {  
    Male = 'MALE',  
    Female = 'FEMALE',  
}  
  
export enum ActivityLevel {  
    Sedentary = 'SEDENTARY',  
    Light = 'LIGHT',  
    Moderate = 'MODERATE',  
    Active = 'ACTIVE',  
    VeryActive = 'VERYACTIVE',  
}
```



بخش کلاس داده های ورودی

در این بخش ابتدا یک Method به نام `Calculable` می سازیم که دارای یک `Method accept` می باشد سپس سه کلاس می سازیم که هر یک از آن ها `Method accept` مربوط به خود را همراه با داده مورد نظر را فراخوانی می کند.



Elements

```
import { ActivityLevel, Gender } from './enums';
import CalculatorVisitor from './visitor.interface';

interface Calculable {
  accept(visitor: CalculatorVisitor): number;
}

export class BMIData implements Calculable {
  constructor(public weight: number, public height: number) {}
  accept(visitor: CalculatorVisitor): number {
    return visitor.visitBMI(this);
  }
}

export class BMRData implements Calculable {
  constructor(
    public weight: number,
    public height: number,
    public age: number,
    public gender: Gender
  ) {}
  accept(visitor: CalculatorVisitor): number {
    return visitor.visitBMR(this);
  }
}

export class TDEEData implements Calculable {
  constructor(public bmr: number, public activityLevel: ActivityLevel) {}
  accept(visitor: CalculatorVisitor): number {
    return visitor.visitTDEE(this);
  }
}
```

بخش Visitor

در این بخش یک Interface می سازیم که نوع داده ورودی Method های آن را از کلاس های داده ای که به صورت جداگانه ساختیم، تعیین می کنیم.



CalculatorVisitor

```
● ● ●  
import { BMIData, BMRData, TDEEData } from './elements';  
  
export default interface CalculatorVisitor {  
    visitBMI(data: BMIData): number;  
    visitBMR(data: BMRData): number;  
    visitTDEE(data: TDEEData): number;  
}
```

بخش Concrete Visitor

در این بخش منطق و فرمول های محاسبه BMI و BMR و TDEE را پیاده سازی می کنیم.



Calculator

```
import { BMIData, BMRData, TDEEData } from './elements';
import { ActivityLevel, Gender } from './enums';
import CalculatorVisitor from './visitor.interface';

export default class Calculator implements CalculatorVisitor {
    visitBMI(data: BMIData): number {
        const { weight, height } = data;
        return weight / (height / 100) ** 2;
    }

    visitBMR(data: BMRData): number {
        const { weight, height, age, gender } = data;
        return gender === Gender.Male
            ? 88.36 + 13.4 * weight + 4.8 * height - 5.7 * age
            : 447.6 + 9.2 * weight + 3.1 * height - 4.3 * age;
    }

    visitTDEE(data: TDEEData): number {
        const bmr = data.bmr;
        const activityMultiplier = [
            [ActivityLevel.Sedentary]: 1.2,
            [ActivityLevel.Light]: 1.375,
            [ActivityLevel.Moderate]: 1.55,
            [ActivityLevel.Active]: 1.725,
            [ActivityLevel.VeryActive]: 1.9,
        ][data.activityLevel];
        return bmr * activityMultiplier;
    }
}
```

بخش FitnessCalculator

در این بخش Component مورد نظر خود را به شکل زیر می سازیم.



```
import { useState } from 'react';
import Calculator from './pattern/concrete-visitor';
import { BMIData, BMRData, TDEEData } from './pattern/elements';
import { ActivityLevel, Gender } from './pattern/enums';

const FitnessCalculator: React.FC = () => {
  const [weight, setWeight] = useState<number>(115);
  const [height, setHeight] = useState<number>(178);
  const [age, setAge] = useState<number>(32);
  const [gender, setGender] = useState<Gender>(Gender.Male);
  const [activityLevel, setActivityLevel] = useState<ActivityLevel>(
    ActivityLevel.Moderate
  );

  const calculator = new Calculator();

  const calculateResults = () => {
    const bmiData = new BMIData(weight, height);
    const bmi = bmiData.accept(calculator);

    const bmrData = new BMRData(weight, height, age, gender);
    const bmr = bmrData.accept(calculator);

    const tdeeData = new TDEEData(bmr, activityLevel);
    const tdee = tdeeData.accept(calculator);

    return { bmr, bmi, tdee };
  };

  const { bmi, bmr, tdee } = calculateResults();

  return (
    <div>
      <p>BMR: {bmr.toFixed(2)}</p>
      <p>BMI: {bmi.toFixed(2)}</p>
      <p>TDEE: {tdee.toFixed(2)}</p>
    </div>
  );
}

export default FitnessCalculator;
```

```
<div>
  <label>
    Weight (kg):
    <input
      type="number"
      value={weight}
      onChange={(e) => setWeight(Number(e.target.value))}>
  </label>
  <br />
  <label>
    Height (cm):
    <input
      type="number"
      value={height}
      onChange={(e) => setHeight(Number(e.target.value))}>
  </label>
  <br />
  <label>
    Age:
    <input
      type="number"
      value={age}
      onChange={(e) => setAge(Number(e.target.value))}>
  </label>
  <br />
  <label>
    Gender:
    <select
      value={gender}
      onChange={(e) => setGender(e.target.value as Gender)}>
      {Object.entries(Gender).map(([key, value]) => (
        <option key={key} value={value}>
          {key}
        </option>
      ))}
    </select>
  <br />
  <label>
    Activity Level:
    <select
      value={activityLevel}
      onChange={(e) => setActivityLevel(e.target.value as ActivityLevel)}>
      {Object.entries(ActivityLevel).map(([key, value]) => (
        <option key={key} value={value}>
          {key}
        </option>
      ))}
    </select>
  </label>
  <h2>Results</h2>
  <p>BMR: {bmr.toFixed(2)}</p>
  <p>BMI: {bmi.toFixed(2)}</p>
  <p>TDEE: {tdee.toFixed(2)}</p>
</div>;
```

طرز استفاده از FitnessCalculator

در پایان به شکل زیر از FitnessCalculator استفاده می کنیم.



VisitorDesignPattern

```
import FitnessCalculator from './FitnessCalculator';

const VisitorDesignPattern = () => {
  return (
    <div>
      <FitnessCalculator />
    </div>
  );
};

export default VisitorDesignPattern;
```