|  |  |
| --- | --- |
| class **Complex** {  **constructor**(re = 0, im = 0) {          this.re = re;          this.im = im;      }  } | class **Vector** {  **constructor**(values = []) {          this.values = [];          values.forEach(elem => this.values.push(elem));      }  } |
|  | class **Matrix** {  **constructor**(values = [[]]) {          this.values = [];          values.forEach((arr, i) => {              this.values[i] = [];              arr.forEach(elem => this.values[i].push(elem));          });      }  } |

class **RealCalculator** {

**get**(a) {

**return** (a instanceof **Matrix** ) ? new **MatrixCalculator**() :

               (a instanceof **Vector** ) ? new **VectorCalculator**() :

               (a instanceof **Complex**) ? new **ComplexCalculator**() : new **RealCalculator**();

    }

**type**(calc, elem, method) {

**if** (elem instanceof **Matrix**) {

**return** calc[method](elem.values.length, elem.values[0][0]);

        } **else** **if** (elem instanceof **Vector**) {

**return** calc[method](elem.values.length, elem.values[0]);

        }

**return** calc[method]();

    }

**add**(a, b) {

**return** a + b;

    }

**sub**(a, b) {

**return** a - b;

    }

**mult**(a, b) {

**return** a \* b;

    }

**div**(a, b) {

**return** a / b;

    }

**prod**(p, a) {

**return** p \* a;

    }

**pow**(a, n) {

**return** Math.pow(a, n);

    }

**one**() {

**return** 1;

    }

**zero**() {

**return** 0;

    }

    /\* unused methods \*/

**module**(a) {

**return** Math.abs(a);

    }

}

class **ComplexCalculator** extends **RealCalculator** {

**add**(a, b) {

**return** new **Complex**(a.re + b.re, a.im + b.im);

    }

**sub**(a, b) {

**return** new **Complex**(a.re - b.re, a.im - b.im);

    }

**mult**(a, b) {

**return** new **Complex**(

            a.re \* b.re - a.im \* b.im,

            a.re \* b.im + a.im \* b.re

        );

    }

**div**(a, b) {

**const** m = Math.pow(b.re, 2) + Math.pow(b.im, 2);

**return** new **Complex**(

            (a.re \* b.re + a.im \* b.im) / m,

            (a.im \* b.re - a.re \* b.im) / m

        );

    }

**prod**(p, a) {

**return** new **Complex**(a.re \* p, a.im \* p);

    }

**pow**(a, n) {

**let** c = this.one();

**for** (**let** i = 0; i < n; i++) {

            c = this.mult(a, c);

        }

**return** c;

    }

**one**() {

**return** new **Complex**(1);

    }

**zero**() {

**return** new **Complex**();

    }

    /\* unused methods \*/

**module**(a) {

**return** Math.sqrt(Math.pow(a.re, 2) + Math.pow(a.im, 2));

    }

}

class **VectorCalculator** extends **RealCalculator** {

**add**(a, b) {

**const** calc = this.get(a.values[0]);

**return** new **Vector**(a.values.map((elem, i) => calc.add(elem, b.values[i])));

    }

**sub**(a, b) {

**const** calc = this.get(a.values[0]);

**return** new **Vector**(a.values.map((elem, i) => calc.sub(elem, b.values[i])));

    }

**mult**(a, b) {

**const** calc = this.get(a.values[0]);

**return** new **Vector**([

            calc.sub(calc.mult(a.values[1], b.values[2]), calc.mult(a.values[2], b.values[1])),

            calc.sub(calc.mult(a.values[2], b.values[0]), calc.mult(a.values[0], b.values[2])),

            calc.sub(calc.mult(a.values[0], b.values[1]), calc.mult(a.values[1], b.values[0]))

        ]);

    }

**prod**(p, a) {

**const** calc = this.get(a.values[0]);

**return** new **Vector**(a.values.map(elem => calc.prod(p, elem)));

    }

**pow**(a, n) {

**let** c = this.one(a.values.length, a.values[0]);

**for** (**let** i = 0; i < n; i++) {

            c = this.mult(a, c);

        }

**return** c;

    }

**one**(length, elem) {

**const** calc = this.get(elem);

**const** values = [];

**for** (**let** i = 0; i < length; i++) {

            values.push(this.type(calc, elem, 'one'));

        }

**return** new **Vector**(values);

    }

**zero**(length, elem) {

**const** calc = this.get(elem);

**const** values = [];

**for** (**let** i = 0; i < length; i++) {

            values.push(this.type(calc, elem, 'zero'));

        }

**return** new **Vector**(values);

    }

    /\* unused methods \*/

**module**(a) {

**return** Math.sqrt(a.values.reduce((S, elem) => S + Math.pow(elem, 2), 0));

    }

**div**() { **return** null; }

}

class **MatrixCalculator** extends **RealCalculator** {

**add**(a, b) {

**const** calc = this.get(a.values[0][0]);

**return** new **Matrix**(a.values.map((arr, i) => arr.map((elem, j) => calc.add(elem, b.values[i][j]))));

    }

**sub**(a, b) {

**const** calc = this.get(a.values[0][0]);

**return** new **Matrix**(a.values.map((arr, i) => arr.map((elem, j) => calc.sub(elem, b.values[i][j]))));

    }

**mult**(a, b) {

**const** calc = this.get(a.values[0][0]);

**const** values = [];

**for** (**let** i = 0; i < a.values.length; i++) {

            values.push([]);

**for** (**let** j = 0; j < a.values[i].length; j++) {

**let** s = calc.zero(null, a.values[0][0]);

**for** (**let** k = 0; k < a.values[i].length; k++) {

                    s = calc.add(s, calc.mult(a.values[k][i], b.values[j][k]));

                }

                values[i][j] = s;

            }

        }

**return** new **Matrix**(values);

    }

**prod**(p, a) {

**const** calc = this.get(a.values[0][0]);

**return** new **Matrix**(a.values.map(arr => arr.map(elem => calc.prod(elem, p))));

    }

**pow**(a, n) {

**let** c = this.one(a.values.length, a.values[0][0]);

**for** (**let** i = 0; i < n; i++) {

            c = this.mult(a, c);

        }

**return** c;

    }

**one**(length, elem) {

**const** calc = this.get(elem);

**const** values = [];

**for** (**let** i = 0; i < length; i++) {

            values.push([]);

**for** (**let** j = 0; j < length; j++) {

                values[i][j] = i === j ? this.type(calc, elem, 'one') : this.type(calc, elem, 'zero');

            }

        }

**return** new **Matrix**(values);

    }

**zero**(length, elem) {

**const** calc = this.get(elem);

**const** values = [];

**for** (**let** i = 0; i < length; i++) {

            values.push([]);

**for** (**let** j = 0; j < length; j++) {

                values[i][j] = this.type(calc, elem, 'zero');

            }

        }

**return** new **Matrix**(values);

    }

    /\* unused methods \*/

**module**() { **return** null; }

**div**() { **return** null; }

}

class **Calculator** extends **RealCalculator** {

**complex**(re, im) {

**return** new **Complex**(re, im);

    }

**vector**(values) {

**return** new **Vector**(values);

    }

**matrix**(values) {

**return** new **Matrix**(values);

    }

**add**(a, b) {

**return** this.get(a).add(a, b);

    }

**sub**(a, b) {

**return** this.get(a).sub(a, b);

    }

**mult**(a, b) {

**return** this.get(a).mult(a, b);

    }

**prod**(p, a) {

**if** (typeof p === 'number') {

**return** this.get(a).prod(p, a);

        }

**return** null;

    }

**pow**(a, n) {

**if** (typeof n === 'number') {

**return** this.get(a).pow(a, n);

        }

**return** null;

    }

**one**(type, elem) {

        type = type ? type : (elem) ? elem.constructor.name : null;

**switch** (type) {

**case** 'Complex': **return** (new **ComplexCalculator**).one();

**case** 'Vector': **return** (new **VectorCalculator**).one(elem.values.length, elem.values[0]);

**case** 'Matrix': **return** (new **MatrixCalculator**).one(elem.values.length, elem.values[0][0]);

**default**: **return** 1;

        }

    }

**zero**(type, elem) {

        type = type ? type : (elem) ? elem.constructor.name : null;

**switch** (type) {

**case** 'Complex': **return** (new **ComplexCalculator**).zero();

**case** 'Vector': **return** (new **VectorCalculator**).zero(elem.values.length, elem.values[0]);

**case** 'Matrix': **return** (new **MatrixCalculator**).zero(elem.values.length, elem.values[0][0]);

**default**: **return** 0;

        }

    }

    /\* unused methods \*/

**module**() { **return** null; }

**div**() { **return** null; }

}

|  |  |
| --- | --- |
| class **Member** {  **constructor**(value = 0, power = 0) {          this.value = value;          this.power = power;      }  } | class **Polynomial** {  **constructor**(poly = []) {          this.poly = poly;          this.poly.sort((a, b) => b.power - a.power);      }  **getValue**(x) {  **const** calc = new **Calculator**();  **return** this.poly.reduce((S, elem) =>              calc.add(S, calc.mult(elem.value, calc.pow(x, elem.power))),              calc.zero(null, x)          );      }  } |

class **PolynomialCalculator** {

**polynomial**(members = []) {

**return** new **Polynomial**(members);

    }

**add**(a, b) {

**const** calc = new **Calculator**();

**const** members = [];

        // заполнить новые члены всеми членами a, сложенными с соответствующими членами b

        a.poly.forEach(elemA => {

**const** member = b.poly.find(elemB => elemB.power === elemA.power);

**if** (member) {

                members.push(new **Member**(calc.add(elemA.value, member.value), elemA.power));

            } **else** {

                members.push(new **Member**(elemA.value, elemA.power));

            }

        });

        // заполнить новые члены всеми оставшимися членами b

        b.poly.forEach(elemB => {

**if** (!members.find(element => element.power === elemB.power)) {

                members.push(new **Member**(elemB.value, elemB.power));

            }

        });

**return** new **Polynomial**(members);

    }

**sub**(a, b) {

**const** calc = new **Calculator**();

**const** members = [];

        a.poly.forEach(elemA => {

**const** member = b.poly.find(elemB => elemB.power === elemA.power);

**if** (member) {

                members.push(new **Member**(calc.sub(elemA.value, member.value), elemA.power));

            } **else** {

                members.push(new **Member**(elemA.value, elemA.power));

            }

        });

        b.poly.forEach(elemB => {

**if** (!members.find(elem => elem.power === elemB.power)) {

                members.push(new **Member**(calc.prod(-1, elemB.value), elemB.power));

            }

        });

**return** new **Polynomial**(members);

    }

**mult**(a, b) {

**const** calc = new **Calculator**();

**let** polynomial = new **Polynomial**();

        a.poly.forEach(elemA => {

**const** members = [];

            b.poly.forEach(elemB => {

                members.push(new **Member**(calc.mult(elemA.value, elemB.value), elemA.power + elemB.power));

            });

            polynomial = this.add(polynomial, new **Polynomial**(members));

        });

**return** polynomial;

    }

}