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

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

    }

**sub**(a, b) {

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

    }

    // векторное произведение

**mult**(a, b) {

**return** new **Vector**([

            a.values[1] \* b.values[2] - a.values[2] \* b.values[1],

            a.values[2] \* b.values[0] - a.values[0] \* b.values[2],

            a.values[0] \* b.values[1] - a.values[1] \* b.values[0]

        ]);

    }

**prod**(p, a) {

**return** new **Vector**(a.values.map(elem => elem \* p));

    }

**pow**(a, n) {

        let c = this.one(a.values.length);

        for (let i = 0; i < n; i++) {

            c = this.mult(a, c);

        }

**return** c;

    }

**one**(length) {

        const values = [];

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

            values.push(1);

        }

**return** new **Vector**(values);

    }

**zero**(length) {

        const values = [];

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

            values.push(0);

        }

**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) {

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

    }

**sub**(a, b) {

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

    }

**mult**(a, b) {

        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 = 0;

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

                    s = s + a.values[k][i] \* b.values[j][k];

                }

                values[i][j] = s;

            }

        }

**return** new **Matrix**(values);

    }

**prod**(p, a) {

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

    }

**pow**(a, n) {

        let c = this.one(a.values.length);

        for (let i = 0; i < n; i++) {

            c = this.mult(a, c);

        }

**return** c;

    }

**one**(length) {

        const values = [];

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

            values.push([]);

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

                values[i][j] = i === j ? 1 : 0;

            }

        }

**return** new **Matrix**(values);

    }

**zero**(length) {

        const values = [];

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

            values.push([]);

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

                values[i][j] = 0;

            }

        }

**return** new **Matrix**(values);

    }

    /\* unused methods \*/

**module**() { **return** null; }

**div**() { **return** null; }

}