1.

var Node = function (name) {

    this.children = [];

    this.name = name;

    }

    Node.prototype = {

    add: function (child) {

    this.children.push(child);

    },

    remove: function (child) {

    var length = this.children.length;

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

    if (this.children[i] === child) {

    this.children.splice(i, 1);

    return;

    }

    }

    },

    getChild: function (i) {

    return this.children[i];

    },

    hasChildren: function () {

    return this.children.length > 0;

    }

    }

    // recursively traverse a (sub)tree

    function traverse(indent, node) {

    console.log(Array(indent++).join("--") + node.name);

    for (var i = 0, len = node.children.length; i < len; i++) {

    traverse(indent, node.getChild(i));

    }

    }

    function run() {

    var tree = new Node("root");

    var left = new Node("left")

    var right = new Node("right");

    var leftleft = new Node("leftleft");

    var leftright = new Node("leftright");

    var rightleft = new Node("rightleft");

    var rightright = new Node("rightright");

    tree.add(left);

    tree.add(right);

    tree.remove(right); // note: remove

    tree.add(right);

    left.add(leftleft);

    left.add(leftright);

    right.add(rightleft);

    right.add(rightright);

    traverse(1, tree);

    }

2.

Behavioral design pattern will be preferred for this problem.

public interface TaxCalculator {

    public abstract void execute();

}

public class Humanity implements TaxCalculator {

    private int basic\_salary;

    public Order(int basic\_salary) {

        this.basic\_salary = basic\_salary;

    }

    @Override

    public void execute() {

        HRA=(10/100)\*basicsalary;

    }

}

public class Logistic implements TaxCalculator {

    private int basic\_salary;

    public Order(int basic\_salary) {

        this.basic\_salary = basic\_salary;

    }

    @Override

    public void execute() {

        HRA=(10/100)\*basicsalary;

    }

}

public class Department {

    public static void main(String[] args) {

        basic\_salary basic\_salary = new basic\_salary();

        Humanity humanity = new Humanity(basic\_salary);

        Logistic logistic = new Logistic(basic\_salary);

        Humanity.execute();

        humanity = new humanity(basic\_salary);

        logistic = new Logistic(basic\_salary);

        Logistic.execute();

    }

}

3.

const arr = [1, 4, 3, 8, 10, 9, 2];

var sum = 0;

for (var number of arr) {

    sum += number;

}

average = sum / arr.length;

const findVariance = (arr = []) => {

   if(!arr.length){

      return 0;

   };

   const sum = arr.reduce((acc, val) => acc + val);

   const { length: num } = arr;

   const median = sum / num;

   let variance = 0;

   arr.forEach(num => {

      variance += ((num - median) \* (num - median));

   });

   variance /= num;

   return variance;

};

console.log(average);

console.log(findVariance(arr))

const getData = async() => {

    var data = average;

    return data;

}

getData().then(data => console.log(data));

const getData2 = async() => {

    var data = findVariance(arr);

    return data;

}

getData2().then(data => console.log(data));

4.

class productId

{

constructor( productId, ProductName,Productprice)

{

this.productId=productId;

this.ProductName=ProductName;

this.Productprice=Productprice;

}

}

let ob1=new productId(7745,qwe,4567);

let ob2=new productId(300,asd,1234);