```
//Class Animal with 4 propertis and 2 methods
class Animal {
    constructor (name, age, color, legs) {
        this.name=name;
        this.age=age;
        this.color=color;
        this.legs=legs;
    }

    get Name() {
        return this.name;
    }

    get Age() {
        return this.age;
    }
    get Sound() {
        return 'EEEEE';
    }
}
//Create a dog and cat
//Override methods
class Dog extends Animal {
    constructor (name, age, color, legs, ppp) {
        super (name, age, color, legs);
        this.ppp=ppp;
    }
    get Sound() {
        return 'Whoof, whoof';
    }
    get DogInfo() {
        return `Name: ${this.name}\nAge: ${this.age}\nColor:
${this.color}\nLegs: ${this.legs}\nPPP: ${this.ppp}`;
    }
}
class Cat extends Animal {
    constructor (name, age, color, legs, breed) {
        super (name, age, color, legs);
        this.breed=breed;
    }
    get Sound () {
        return 'Miaow, miaow';
    }
}
//Class PersonalAccount
class PersonalAccount {
    constructor (firstName, lastName) {
        this.firstName=firstName;
        this.lastName=lastName;
        this.incomes=new Set();
        this.expenses=new Set();
    }
    get totalIncome() {
        return [...this.incomes].reduce((total, income) => total +
income.amount, 0);
    }
    get totalExpense() {
        return [...this.expenses].reduce((total, expense) => total +
expense.amount, 0);
    }
    addIncome (description, amount) {
        this.incomes.add({description, amount});
    }
    addExpense (description, amount) {
        this.expenses.add({description, amount});
    }
    get accountBalance () {
        return (this.totalIncome-this.totalExpense);
    }
}
//Classes to calculate measurements
class Statistics {

    static count(ages) {
        return ages.length;
    }

    static sum(ages) {
        return ages.reduce((acc, n) => acc + n, 0);
    }

    static min(ages) {
        const sorted = ages.slice().sort((a, b) => a - b);
        return sorted[0];
    }

    static max(ages) {
        const sorted = ages.slice().sort((a, b) => b - a);
        return sorted[0];
    }

    static range(ages) {
        return this.max(ages) - this.min(ages);
    }

    static mean(ages) {
        return Math.floor(this.sum(ages) / this.count(ages))+1;
    }

    static median(ages) {
        const sorted = ages.slice().sort((a, b) => a - b);
        const middle = Math.floor(sorted.length / 2);

        if (sorted.length % 2 === 0) {
            return (sorted[middle - 1] + sorted[middle]) / 2;
        } else {
            return sorted[middle];
        }
    }

    static mode(ages) {
        const frequency = {};
        let maxCount = 0;
        let mode = null;

        for (const num of ages) {
            frequency[num] = (frequency[num] || 0) + 1;
            if (frequency[num] > maxCount) {
                maxCount = frequency[num];
                mode = num;
            }
        }
        return {mode, maxCount };
    }

    static describe(ages) {
        return `Count: ${this.count(ages)}\nSum: ${this.sum(ages)}\nMin: 
${this.min(ages)}\nMax: ${this.max(ages)}\nRange: ${this.range(ages)}\nMean: 
${this.mean(ages)}\nMedian: ${this.median(ages)}\nMode: 
${JSON.stringify(this.mode(ages))}`;
    }
}
```