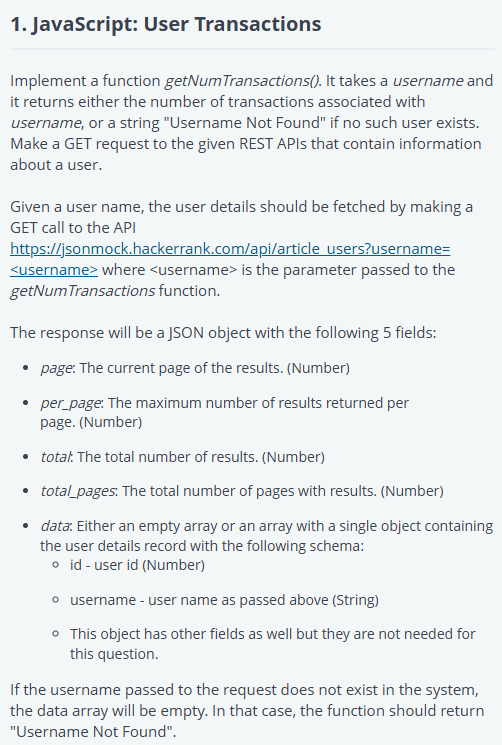
# JavaScript: User Transactions



'use strict';

const fs = require('fs');

const https = require('https');

process.stdin.resume();

process.stdin.setEncoding('utf-8');

let inputString = '';

let currentLine = 0;

process.stdin.on('data', function(inputStdin) {

      inputString += inputStdin;

});

process.stdin.on('end', function() {

    inputString = inputString.split('\n');

    main();

});

function readLine() {

      return inputString[currentLine++];

}

async function getNumTransactions(username) {

    // write your code here

    // API endpoint: https://jsonmock.hackerrank.com/api/article\_users?username=<username>

    // API endpoint: https://jsonmock.hackerrank.com/api/transactions?&userId=<userId>

    // First API to fetch user details

    const userDetailsUrl = `https://jsonmock.hackerrank.com/api/article\_users?username=${username}`;

    const userDetails = await fetchData(userDetailsUrl);

    if (userDetails.data.length === 0) {

        return "Username Not Found";

    }

    const userId = userDetails.data[0].id;

    // Second API to fetch user transactions

    const transactionsUrl = `https://jsonmock.hackerrank.com/api/transactions?userId=${userId}`;

    const transactions = await fetchData(transactionsUrl);

    return transactions.total;

}

function fetchData(url) {

    return new Promise((resolve, reject) => {

        https.get(url, (res) => {

            let data = '';

            res.on('data', (chunk) => {

                data += chunk;

            });

            res.on('end', () => {

                resolve(JSON.parse(data));

            });

        }).on('error', (e) => {

            reject(e);

        });

    });

}

async function main() {

    const ws = fs.createWriteStream(process.env.OUTPUT\_PATH);

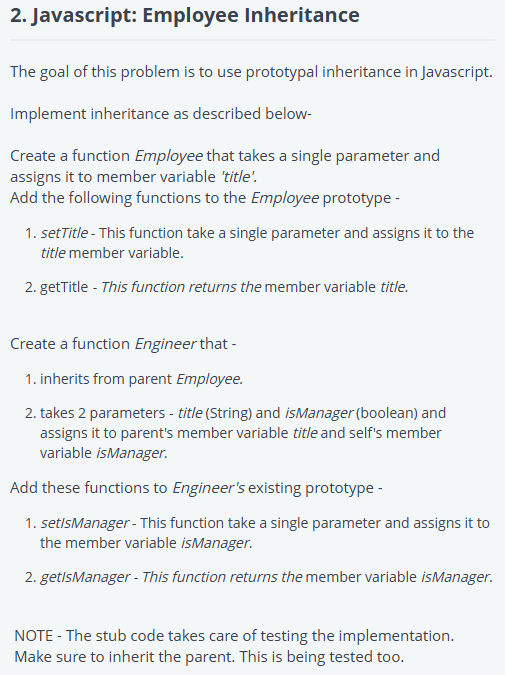
    const username = readLine().trim();

    const result = await getNumTransactions(username);

    ws.write(result.toString());

}

# JavaScript: Employee Inheritance



'use strict';

const fs = require('fs');

process.stdin.resume();

process.stdin.setEncoding("ascii");

let inputString = "";

let currentLine = 0;

process.stdin.on("data", function (chunk) {

    inputString += chunk;

});

process.stdin.on("end", function () {

    inputString = inputString.split('\n');

    main();

});

function readLine() {

  return inputString[currentLine++];

}

function Employee(title) {

    this.title = title;

}

Employee.prototype.setTitle = function(newTitle) {

    this.title = newTitle;

};

Employee.prototype.getTitle = function() {

    return this.title;

};

function Engineer(title, isManager) {

    Employee.call(this, title);

    this.isManager = isManager;

}

Engineer.prototype = Object.create(Employee.prototype);

Engineer.prototype.constructor = Engineer;

Engineer.prototype.setIsManager = function(newIsManager) {

    this.isManager = newIsManager;

};

Engineer.prototype.getIsManager = function() {

    return this.isManager;

};

function main() {

    const ws = fs.createWriteStream(process.env.OUTPUT\_PATH);

    var inputs = readLine().split(' ');

    var engineerObject = new Engineer(inputs[0], inputs[1].toLowerCase() === 'true');

    ws.write(`Initial Employee Profile - Title is ${engineerObject.getTitle()}. ${engineerObject.getIsManager() ? 'Is' : 'Is not'} a Manager\n`)

    engineerObject.setTitle(readLine());

    engineerObject.setIsManager(readLine().toLowerCase() === 'true');

    ws.write(`Final Employee Profile - Title is ${engineerObject.getTitle()}. ${engineerObject.getIsManager() ? 'Is' : 'Is not'} a Manager\n`)

    ws.write(`Engineer.prototype has property setTitle: ${Engineer.prototype.hasOwnProperty('setTitle')}\n`);

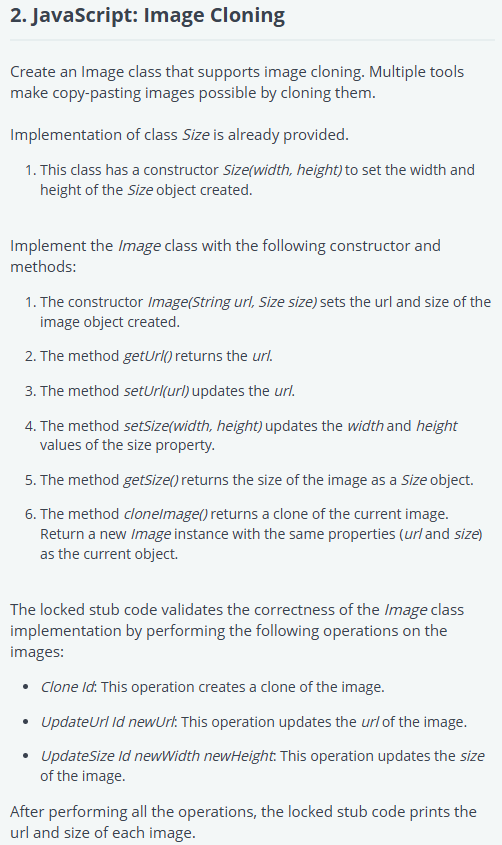
    ws.write(`Engineer.prototype has property getTitle: ${Engineer.prototype.hasOwnProperty('getTitle')}\n`);

    ws.write(`Engineer.prototype has property setIsManager: ${Engineer.prototype.hasOwnProperty('setIsManager')}\n`);

    ws.write(`Engineer.prototype has property getIsManager: ${Engineer.prototype.hasOwnProperty('getIsManager')}\n`);

}

# JavaScript: Image Cloning



'use strict';

const fs = require('fs');

process.stdin.resume();

process.stdin.setEncoding("ascii");

let inputString = "";

let currentLine = 0;

process.stdin.on("data", function (chunk) {

    inputString += chunk;

});

process.stdin.on("end", function () {

    inputString = inputString.split('\n');

    main();

});

function readLine() {

  return inputString[currentLine++];

}

class Size {

    constructor(width, height) {

        this.width = width;

        this.height = height;

    }

}

class Image {

    constructor(url, size) {

        this.url = url;

        this.size = size;

    }

    getUrl() {

        return this.url;

    }

    setUrl(url) {

        this.url = url;

    }

    setSize(width, height) {

        this.size.width = width;

        this.size.height = height;

    }

    getSize() {

        return this.size;

    }

    cloneImage() {

        return new Image(this.url, new Size(this.size.width, this.size.height));

    }

}

function main() {

    const ws = fs.createWriteStream(process.env.OUTPUT\_PATH);

    let images = [];

    let numberOfImages = parseInt(readLine().trim());

    while (numberOfImages-- > 0) {

        let inputs = readLine().trim().split(' ');

        images.push(new Image(inputs[0], new Size(parseInt(inputs[1]), parseInt(inputs[2]))));

    }

    let numberOfOperations = parseInt(readLine().trim());

    while (numberOfOperations-- > 0) {

        let inputs = readLine().trim().split(' ');

        const image = images[parseInt(inputs[1]) - 1];

        const operation = inputs[0];

        switch(operation) {

            case 'Clone':

                images.push(image.cloneImage());

                break;

            case 'UpdateUrl':

                image.setUrl(inputs[2]);

                break;

            case 'UpdateSize':

                image.setSize(parseInt(inputs[2]), parseInt(inputs[3]));

                break;

            default:

                break;

        }

    }

    images.forEach((img) => {

        const size = img.getSize();

        ws.write(`${img.getUrl()} ${size.width} ${size.height}\n`);

    })

}