|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 1**

|  |  |
| --- | --- |
| **Тема:** Создание и использование структур на JavaScript  **Дисциплина:** Архитектура ЭВМ  **Студент:** Платонова Ольга  **Группа:** ИУ7-55Б |  |

Москва.

2020 г.

**Цель работы:** знакомство с языком JavaScript, использование простейших структур, таких как массив, список; написание классов и их методов; написание составных классов.

**Модуль 1**

**Задание 1.**

Создать хранилище в оперативной памяти для хранения информации о детях. Необходимо хранить информацию о ребенке: фамилия и возраст. Необходимо обеспечить уникальность фамилий детей.

Реализовать функции:

* CREATE READ UPDATE DELETE для детей в хранилище
* Получение среднего возраста детей
* Получение информации о самом старшем ребенке
* Получение информации о детях, возраст которых входит в заданный отрезок
* Получение информации о детях, фамилия которых начинается с заданной буквы
* Получение информации о детях, фамилия которых длиннее заданного количества символов
* Получение информации о детях, фамилия которых начинается с гласной буквы

**Листинг 1.**

"use strict";

class Children

{

constructor(surname, age) {

this.surname = surname;

this.age = age;

}

renderFields() {

console.log(this.key + " " + this.surname + " " + this.age);

}

}

let ChildDic = new Map();

ChildDic.set("Иванов", 4);

ChildDic.set("Петров", 2);

ChildDic.set("Сидоров", 6);

ChildDic.set("Котова", 8);

create("Кнопочкин", 5);

output();

read("Петров");

update("Иванов", 7);

read("Иванов");

deleteChild("Сидоров");

console.log();

output();

average();

oldestChild();

readInterval(3, 6);

readLetter("И");

readLength(7);

readFirstLetter();

function output()

{

for (let [key, value] of ChildDic) {

console.log(key + " " + value);

}

console.log("");

}

function create(name, age)

{

let fr = 0;

for (let [key, value] of ChildDic) {

if (key === name) {

fr = 1;

break;

}

}

if (!fr) {

ChildDic.set(name, age);

} else {

console.log("Repeated surname");

}

}

function read(name)

{

let fr = 0;

for (let [key, value] of ChildDic) {

if (key === name) {

fr = 1;

console.log("Child read: " + key + " " + value);

break;

}

}

if (!fr) {

console.log("No such child");

}

}

function update(name, age)

{

let fr = 0;

for (let [key, value] of ChildDic) {

if (key === name) {

fr = 1;

ChildDic.set(name, age);

break;

}

}

if (!fr) {

console.log("No such child");

}

}

function deleteChild(name)

{

let fr = 0;

for (let [key, value] of ChildDic) {

if (key === name) {

fr = 1;

ChildDic.delete(key);

break;

}

}

if (!fr) {

console.log("No such child");

}

}

function average()

{

let res = 0;

for (let [key, value] of ChildDic) {

res += value;

}

console.log("Average age: " + res / ChildDic.size);

}

function oldestChild()

{

let nameOld;

let ageOld = 0;

for (let [key, value] of ChildDic) {

if (value >= ageOld) {

nameOld = key;

ageOld = value;

}

}

console.log("Oldest child: " + nameOld + " " + ageOld);

}

function readInterval(a, b)

{

console.log("\nAge in interval (3; 6):");

for (let [key, value] of ChildDic) {

if (value >= a && value <= b) {

console.log("\t" + key + " " + value);

}

}

}

function readLetter(a)

{

console.log("\nSurname with letter 'И':");

for (let [key, value] of ChildDic) {

if (key.charAt(0) === a) {

console.log("\t" + key + " " + value);

}

}

}

function readLength(a)

{

console.log("\nSurname is 7 letter long:");

for (let [key, value] of ChildDic) {

let len = key.length;

if (len > a) {

console.log("\t" + key + " " + value);

}

}

}

function readFirstLetter()

{

console.log("\nSurname starts with a vowel:");

let vowel = 'УЕЫАОЭЯИЮ';

for (let [key, value] of ChildDic) {

let c = key.charAt(0);

if (vowel.includes(c)) {

console.log("\t" + key + " " + value);

}

}

}

**Задание 2.**

Создать хранилище в оперативной памяти для хранения информации о студентах. Необходимо хранить информацию о студенте: название группы, номер студенческого билета, оценки по программированию. Необходимо обеспечить уникальность номеров студенческих билетов.

Реализовать функции:

* CREATE READ UPDATE DELETE для студентов в хранилище
* Получение средней оценки заданного студента
* Получение информации о студентах в заданной группе
* Получение студента, у которого наибольшее количество оценок в заданной группе
* Получение студента, у которого нет оценок

**Листинг 2.**

**"use strict";**

class Student {

constructor(id, group, marks) {

this.Id = id;

this.Group = group;

this.Marks = marks;

}

output() {

console.log("Student " + this.Id + "\n\tGroup: " + this.Group + "\n\t" +

"Marks: " + this.Marks);

}

}

let s1 = new Student(1, 5, [4, 5, 4, 4]);

let s2 = new Student(8, 5, [3, 4, 3, 4, 3]);

let s3 = new Student(3, 4, [5, 5, 4, 5]);

let s4 = new Student(7, 1, [2, 4, 3]);

let s5 = new Student(2, 5, [5]);

let s6 = new Student(4, 1, []);

let List = [s1, s2, s3, s4, s5, s6];

createStudent(5, 2, [3, 3, 5, 4, 2]);

listOutput();

readStudent(3);

updateStudent(8, 2, [3, 4, 3, 4, 3]);

deleteStudent(3);

averageStudentMark(8);

studentsFromGroup(5);

studentWithLargestNumMarks();

studentsWithoutMarks();

function listOutput()

{

for (let key in List) {

List[key].output();

}

console.log();

}

function output(msg, id)

{

console.log(msg);

List[id].output();

}

function checkExistance(fe)

{

if (fe) {

console.log("No such student exists");

}

}

function createStudent(id, group, marks)

{

let fe = 0;

for (let key in List) {

if (List[key].Id === id) {

fe = 1;

console.log("Id already exissts");

break;

}

}

if (!fe) {

let s = new Student(id, group, marks);

List.push(s);

}

}

function readStudent(id)

{

let fe = 1;

for (let key in List) {

if (List[key].Id === id) {

fe = 0;

output("Found student:", key);

break;

}

}

checkExistance(fe);

}

function updateStudent(id, group, marks)

{

let fe = 1;

for (let key in List) {

if (List[key].Id === id) {

fe = 0;

let s = new Student(id, group, marks);

List[key] = s;

output("\nUpdated student:", key);

break;

}

}

checkExistance(fe);

}

function deleteStudent(id)

{

let fe = 1;

for (let key in List) {

if (List[key].Id === id) {

fe = 0;

output("\nDeleted student:", key);

List.splice(key, 1);

break;

}

}

checkExistance(fe);

}

function averageStudentMark(id)

{

let fe = 1;

for (let key in List) {

if (List[key].Id === id) {

fe = 0;

console.log("\nAverage mark: " + average(List[key].Marks));

break;

}

}

checkExistance(fe);

}

function average(arr)

{

let sum = 0;

for (let i in arr) {

sum += arr[i];

}

return sum / arr.length;

}

function studentsFromGroup(group)

{

console.log("\nStudents from group " + group + ":");

let fe = 1;

for (let key in List) {

if (List[key].Group == group) {

fe = 0;

List[key].output();

}

}

checkExistance(fe);

}

function studentWithLargestNumMarks()

{

let maxKey = 0, maxNum = 0;

for (let key in List) {

if (List[key].Marks.length > maxNum) {

maxNum = List[key].Marks.length;

maxKey = key;

}

}

output("\nStudent with the largest num of marks:", maxKey);

}

function studentsWithoutMarks()

{

console.log("\nStudents without marks:");

for (let key in List) {

if (List[key].Marks.length == 0) {

List[key].output();

}

}

}

**Задание 3.**

Создать хранилище в оперативной памяти для хранения точек. Неоходимо хранить информацию о точке: имя точки, позиция X и позиция Y. Необходимо обеспечить уникальность имен точек.

Реализовать функции:

* CREATE READ UPDATE DELETE для точек в хранилище
* Получение двух точек, между которыми наибольшее расстояние
* Получение точек, находящихся от заданной точки на расстоянии, не превышающем заданную константу
* Получение точек, находящихся выше / ниже / правее / левее заданной оси координат
* Получение точек, входящих внутрь заданной прямоугольной зоны

**Листинг 3.**

**"use strict";**

class Point {

constructor(name, x, y) {

this.Name = name;

this.X = x;

this.Y = y;

}

output() {

console.log("Point " + this.Name + "\n\t(" + this.X + "; " + this.Y + ")");

}

}

let p1 = new Point('A', 1, 0);

let p2 = new Point('E', 3, 6);

let p3 = new Point('B', -8, -5);

let p4 = new Point('S', 4, 2);

let p5 = new Point('N', 5, -1);

let p6 = new Point('O', 0, 0);

let List = [p1, p2, p3, p4, p5, p6];

createPoint('T', 4, 10);

listOutput();

readPoint('S');

updatePoint('B', -8, -2);

deletePoint('N');

longestDistance();

listOutput();

givenDistance('O', 6);

givenDirection('U');

let pa = new Point('A', -2, -2);

let pb = new Point('B', -2, 3);

let pc = new Point('C', 2, 3);

let pd = new Point('D', 2, -2);

insideRectangle(pa, pb, pc, pd);

function listOutput()

{

for (let key in List) {

List[key].output();

}

console.log();

}

function output(msg, id)

{

console.log(msg);

List[id].output();

}

function checkExistance(fe)

{

if (fe) {

console.log("No such point exists");

}

}

function dist(p1, p2)

{

let res = (p2.X - p1.X) \* (p2.X - p1.X);

res += (p2.Y - p1.Y) \* (p2.Y - p1.Y);

return Math.sqrt(res);

}

function createPoint(name, x, y)

{

let fe = 0;

for (let key in List) {

if (List[key].Name === name) {

fe = 1;

console.log("Name already exists");

break;

}

}

if (!fe) {

let p = new Point(name, x, y);

List.push(p);

}

}

function readPoint(name)

{

let fe = 1;

for (let key in List) {

if (List[key].Name === name) {

fe = 0;

output("Found point:", key);

break;

}

}

checkExistance(fe);

}

function updatePoint(name, x, y)

{

let fe = 1;

for (let key in List) {

if (List[key].Name === name) {

fe = 0;

let p = new Point(name, x, y);

List[key] = p;

output("\nUpdated point:", key);

break;

}

}

checkExistance(fe);

}

function deletePoint(name)

{

let fe = 1;

for (let key in List) {

if (List[key].Name === name) {

fe = 0;

output("\nDeleted point:", key);

List.splice(key, 1);

break;

}

}

checkExistance(fe);

}

function longestDistance()

{

let p1, p2, maxDist = 0;

for (let i = 0; i < List.length - 1; i++) {

for (let j = i + 1; j < List.length; j++) {

let curDist = dist(List[i], List[j]);

if (curDist > maxDist) {

maxDist = curDist;

p1 = i; p2 = j;

}

}

}

console.log("\nThe longest distance:");

List[p1].output(); List[p2].output();

}

function findPoint(name)

{

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

if (List[i].Name == name) {

return i;

}

}

return -1;

}

function givenDistance(name, givenDist)

{

console.log("\nThe given distance:");

let givenPoint = findPoint(name);

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

if (i == givenPoint) {

continue;

}

let curDist = dist(List[i], List[givenPoint]);

if (curDist <= givenDist) {

List[i].output();

}

}

}

function checkUp()

{

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

if (List[i].Y > 0) {

List[i].output();

}

}

}

function checkDown()

{

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

if (List[i].Y < 0) {

List[i].output();

}

}

}

function checkLeft()

{

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

if (List[i].X < 0) {

List[i].output();

}

}

}

function checkRight()

{

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

if (List[i].X > 0) {

List[i].output();

}

}

}

function givenDirection(dir)

{

console.log("\nPoints " + dir);

if (dir == 'U') {

checkUp();

}

if (dir == 'D') {

checkDown();

}

if (dir == 'L') {

checkLeft();

}

if (dir == 'R') {

checkRight();

}

}

function mult(idx, pa, pb)

{

return ((pb.X - pa.X) \* (List[idx].Y - pa.Y) - (pb.Y - pa.Y) \* (List[idx].X - pa.X));

}

function insideRectangle(pa, pb, pc, pd)

{

console.log("\nPoints inside the rectangle:");

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

let p1 = mult(i, pa, pb);

let p2 = mult(i, pb, pc);

let p3 = mult(i, pc, pd);

let p4 = mult(i, pd, pa);

if ((p1 < 0 && p2 < 0 && p3 < 0 && p4 < 0) ||

(p1 > 0 && p2 > 0 && p3 > 0 && p4 > 0)) {

List[i].output();

}

}

}

**Модуль 2**

**Задание 1.**

Создать класс Точка. Добавить классу точка Точка метод инициализации полей и метод вывода полей на экран. Создать класс Отрезок. У класса Отрезок должны быть поля, являющиеся экземплярами класса Точка. Добавить классу Отрезок метод инициализации полей, метод вывода информации о полях на экран, а так же метод получения длины отрезка.

**Листинг 1.**

"use strict";

class Point {

constructor(x, y) {

this.X = x;

this.Y = y;

}

output() {

console.log("Point: (" + this.X + "; " + this.Y + ")");

}

}

class Segment {

constructor() {

this.PBegin = null;

this.PEnd = null;

}

initSegment(pBegin, pEnd) {

this.PBegin = pBegin;

this.PEnd = pEnd;

}

output() {

console.log("\nSegment:");

this.PBegin.output();

this.PEnd.output();

}

getLength() {

let len = (this.PEnd.X - this.PBegin.X) \* (this.PEnd.X - this.PBegin.X);

len += (this.PEnd.Y - this.PBegin.Y) \* (this.PEnd.Y - this.PBegin.Y);

return Math.sqrt(len);

}

}

let point = new Point(2, 3);

point.output();

let pa = new Point(-4, 1);

let pb = new Point(4, 1);

let ab = new Segment();

ab.initSegment(pb, pa);

ab.output();

**Задание 2.**

Создать класс Треугольник. Класс Треугольник должен иметь поля, хранящие длины сторон треугольника.

Реализовать следующие методы:

* Метод инициализации полей
* Метод проверки возможности существования треугольника с такими сторонами
* Метод получения периметра треугольника
* Метод получения площади треугольника
* Метод для проверки факта: является ли треугольник прямоугольным

**Листинг 2.**

"use strict";

class Triangle {

constructor(len1, len2, len3) {

this.initLen(len1, len2, len3);

}

initLen(len1, len2, len3) {

this.A = len1;

this.B = len2;

this.C = len3;

}

isExist() {

if (this.A > 0 && this.B > 0 && this.C > 0 &&

(this.A + this.B) > this.C &&

(this.A + this.C) > this.B &&

(this.B + this.C) > this.A) {

return true;

}

return false;

}

getPerimeter() {

return (this.A + this.B + this.C);

}

getSquare() {

let p = this.getPerimeter() / 2;

let sqr = p \* (p - this.A) \* (p - this.B) \* (p - this.C);

return Math.sqrt(sqr);

}

isRight() {

let maxLen = max(this.A, this.B, this.C);

let midLen = mid(this.A, this.B, this.C);

let minLen = min(this.A, this.B, this.C);

if (minLen \* minLen + midLen \* midLen === maxLen \* maxLen) {

return true;

}

return false;

}

output() {

console.log("Triangle: " + this.A + " " + this.B + " " + this.C);

}

}

let tr = new Triangle(1, 4, 2);

tr.initLen(4, 5, 3);

tr.output();

if (tr.isExist()) {

console.log("Perimetr: " + tr.getPerimeter());

console.log("Square: " + tr.getSquare());

}

console.log(tr.isRight());

function max(a, b, c) {

let res = (a > b) ? a : b;

return ((c > res) ? c : res);

}

function min(a, b, c) {

let res = (a > b) ? b : a;

return ((c < res) ? c : res);

}

function mid(a, b, c) {

let maxL = max(a, b, c);

let minL = min(a, b, c);

if (a != maxL && a != minL) {

return a;

}

if (b != maxL && b != minL) {

return b;

}

return c;

}

**Задание 3.**

Реализовать программу, в которой происходят следующие действия:

Происходит вывод целых чисел от 1 до 10 с задержками в 2 секунды.

После этого происходит вывод от 11 до 20 с задержками в 1 секунду.

Потом опять происходит вывод чисел от 1 до 10 с задержками в 2 секунды.

После этого происходит вывод от 11 до 20 с задержками в 1 секунду.

Это должно происходить циклически.

**Листинг 3.**

"use strict";

let *i* = 1;

let *end1* = 10, *end2* = 20;

let *tmr1* = setInterval(function cntr() {

*console*.log(*i*);

if (*i* == *end1*) {

let *tmr2* = setInterval(function() {

*console*.log(*i*);

if (*i* == *end2*) {

*i* = 1;

clearInterval(*tmr2*);

cntr();

}

*i*++;

}, 1000);

}

*i*++;

}, 2000);

**Вывод**

В ходе этой лабораторной работы мы познакомились с языком JavaScript, использовали простейшие структуры, такие как массив, список; реализовали написание классов и их методов, написание составных классов.