# Solutions-Lab: Classes

# Classes

# 1.Rectangle

class Rectangle{

constructor(width, height, color){

this.width = width,

this.height = height,

this.color = color

}

calcArea(){

return this.width \* this.height;

}

}

let rectangle = new Rectangle(4, 5, 'red')

console.log(rectangle.width)

console.log(rectangle.height)

console.log(rectangle.color)

console.log(rectangle.calcArea())

# 2.Person

class Person{

constructor(firstName, lastName, age, email){

this.firstName = firstName,

this.lastName = lastName,

this.age = age,

this.email = email

}

toString(){

return `${this.firstName} ${this.lastName} (age: ${this.age}, email: ${this.email})`;

}

}

let person = new Person('Maria', 'Petrova', 22, 'mp@yahoo.com')

console.log(person.toString())

# 3.Get Persons

function getPersons(){

class Person{

constructor(firstName, lastName, age, email){

this.firstName = firstName,

this.lastName = lastName,

this.age = age,

this.email = email

}

toString(){

return `${this.firstName} ${this.lastName} (age: ${this.age}, email: ${this.email})`;

}

}

return [new Person('Maria', 'Petrova', 22, 'mp@yahoo.com'), new Person('SoftUni'), new Person('Stephan', 'Nikolov', 25), new Person('Peter', 'Kolev', 24, 'ptr@gmail.com')];

}

console.log('' + getPersons())

console.log(getPersons())

# 4.Circle

class Circle{

constructor(radius){

this.radius = radius

}

get diameter(){

return 2 \* this.radius;

}

set diameter(diameter){

this.radius = diameter / 2;

}

get area(){

return Math.PI \* this.radius \*\* 2;

}

}

let c = new Circle(2);

console.log(`Radius: ${c.radius}`);

console.log(`Diameter: ${c.diameter}`);

console.log(`Area: ${c.area}`);

c.diameter = 1.6;

console.log(`Radius: ${c.radius}`);

console.log(`Diameter: ${c.diameter}`);

console.log(`Area: ${c.area}`);

## 5.Point Distance

class Point{

constructor(x, y){

this.x = x,

this.y = y

}

static distance(point1, point2){

return Math.sqrt((point1.x - point2.x) \*\* 2 + (point1.y - point2.y) \*\* 2);

}

}

let p1 = new Point(5, 5);

let p2 = new Point(9, 8);

console.log(Point.distance(p1, p2));

## 6.Cards

let madeCardAndSuits = (function(){

let Suits = {

SPADES: '\u2660',

HEARTS: '\u2665',

DIAMONDS: '\u2666',

CLUBS: '\u2663'

}

let Faces = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A'];

class Card{

constructor(face, suit){

this.face = face;

this.suit = suit;

}

get face() {

return this.\_face;

}

set face(face){

if (!Faces.includes(face)){

throw new Error(`Invalid face ${face}`);

}

this.\_face = face;

}

get suit(){

return this.\_suit;

}

set suit(suit){

if(!Object.keys(Suits).map(s => Suits[s]).includes(suit)){

throw new Error(`Invalid suit ${suit}`);

}

this.\_suit = suit;

}

}

return {

Suits: Suits,

Card: Card

}

}())

let Card = madeCardAndSuits.Card;

console.log(Card)

let Suits = madeCardAndSuits.Suits;

console.log(Suits)

let card = new Card('Q', Suits.CLUBS);

console.log(card)

console.log(card.face)

console.log(card.suit)

card.face = 'A';

card.suit = Suits.DIAMONDS;

console.log(card.face)

console.log(card.suit)

let card1 = new Card('1', Suits.DIAMONDS);//Error

|  |
| --- |
| (function () { |
|  | let Suits = { |
|  | CLUBS: "\u2663", // ♣ |
|  | DIAMONDS: "\u2666", // ♦ |
|  | HEARTS: "\u2665", // ♥ |
|  | SPADES: "\u2660" // ♠ |
|  | }; |
|  | let Faces = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']; |
|  | class Card { |
|  | constructor(face, suit) { |
|  | this.face = face; |
|  | this.suit = suit; |
|  | } |
|  |  |
|  | get face() { |
|  | return this.\_face; |
|  | } |
|  | set face(f) { |
|  | if (!Faces.includes(f)) { |
|  | throw new Error(`Invalid face ${f}`); |
|  | } |
|  | this.\_face = f; |
|  | } |
|  |  |
|  | get suit() { |
|  | return this.\_suit; |
|  | } |
|  | set suit(s) { |
|  | if (!Object.keys(Suits).map(k => Suits[k]).includes(s)) { |
|  | throw new Error(`Invalid suit ${s}`); |
|  | } |
|  | this.\_suit = s; |
|  | } |
|  | } |
|  | return { |
|  | Suits, |
|  | Card |
|  | } |
|  | }()) |

|  |
| --- |
| (() => { |
|  | let Suits = { |
|  | CLUBS: "\u2663", |
|  | DIAMONDS: "\u2666", |
|  | HEARTS: "\u2665", |
|  | SPADES: "\u2660" |
|  | }; |
|  |  |
|  | let Faces = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']; |
|  |  |
|  | class Card{ |
|  | constructor(face, suit){ |
|  | this.face = face; |
|  | this.suit = suit; |
|  | } |
|  |  |
|  | get face(){ |
|  | return this.\_face; |
|  | } |
|  | set face(face){ |
|  | if(! Faces.includes(face)){ |
|  | throw new Error("Invalid card face: " + face); |
|  | } |
|  | this.\_face = face; |
|  | } |
|  |  |
|  | get suit(){ |
|  | return this.\_suit; |
|  | } |
|  | set suit(suit){ |
|  | if(! Object.keys(Suits).map(k => Suits[k]).includes(suit)){ |
|  | throw new Error("Invalid card suit: " + suit); |
|  | } |
|  | this.\_suit = suit; |
|  | } |
|  |  |
|  | toString(){ |
|  | return `${this.face}${this.suit}`; |
|  | } |
|  | } |
|  | return {Suits, Card}; |
|  | } |
|  | )(); |

# Unit testing on Classes

## 7.String Builder

class StringBuilder {

constructor(string) {

if (string !== undefined) {

StringBuilder.\_vrfyParam(string);

this.\_stringArray = Array.from(string);

} else {

this.\_stringArray = [];

}

}

append(string) {

StringBuilder.\_vrfyParam(string);

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

this.\_stringArray.push(string[i]);

}

}

prepend(string) {

StringBuilder.\_vrfyParam(string);

for(let i = string.length - 1; i >= 0; i--) {

this.\_stringArray.unshift(string[i]);

}

}

insertAt(string, startIndex) {

StringBuilder.\_vrfyParam(string);

this.\_stringArray.splice(startIndex, 0, ...string);

}

remove(startIndex, length) {

this.\_stringArray.splice(startIndex, length);

}

static \_vrfyParam(param) {

if (typeof param !== 'string') throw new TypeError('Argument must be string');

}

toString() {

return this.\_stringArray.join('');

}

}

module.exports = {StringBuilder};

let expect = require('chai').expect;

let StringBuilder = require('../07StringBuilder').StringBuilder;

//In Judge must be paste without this above

describe('StringBuilder', function(){

let builder;

it('empty initialization should does not throw', function(){

let emptyStringBuilderInitialization = () => builder = new StringBuilder();

expect(emptyStringBuilderInitialization).to.not.throw();

});

it('initialization with parameters should does not throw', function(){

let stringBuilderInitializationWithPrameters = () => builder = new StringBuilder('Parameter');

expect(stringBuilderInitializationWithPrameters).to.not.throw();

});

it('should throw with invalid constructor parameter', function(){

let initializationWithInvalidConstructorParameter = () => builder = new StringBuilder(8);

expect(initializationWithInvalidConstructorParameter).to.throw();

});

describe('StringBuilder initialization with empty constructor', function(){

beforeEach(function(){

builder = new StringBuilder();

});

it('should has all properties', function(){

expect(builder.hasOwnProperty('\_stringArray')).to.equal(true, 'Missing \_stringArray property!');

});

it('should has all functions attached to prototype', function(){

expect(Object.getPrototypeOf(builder).hasOwnProperty('append')).to.equal(true, 'Missing append function!');

expect(Object.getPrototypeOf(builder).hasOwnProperty('prepend')).to.equal(true, 'Missing prepend function!');

expect(Object.getPrototypeOf(builder).hasOwnProperty('insertAt')).to.equal(true, 'Missing insertAt function!');

expect(Object.getPrototypeOf(builder).hasOwnProperty('remove')).to.equal(true, 'Missing remove function!');

expect(Object.getPrototypeOf(builder).hasOwnProperty('toString')).to.equal(true, 'Missing toString function!');

});

it('should initialize data to an empty array', function(){

expect(builder.\_stringArray instanceof Array).to.equal(true, 'Data must be of type array!');

expect(builder.\_stringArray.length).to.equal(0, 'Data array must be initialized empty!');

});

});

describe('StringBuilber initialization with constructor with parameters', function(){

beforeEach(function(){

let startString = 'Parameter';

builder = new StringBuilder(startString);

});

it('should initialize data to a string array', function(){

expect(builder.\_stringArray instanceof Array).to.equal(true, 'Data must be of type array!');

let expectedArray = Array.from('Parameter');

compareArray(builder.\_stringArray, expectedArray);

});

it('should append correctly', function(){

let expectedArray = Array.from('Parameter End');

builder.append(' End');

compareArray(builder.\_stringArray, expectedArray);

});

it('should prepend correctly', function(){

let expectedArray = Array.from('Start Parameter');

builder.prepend('Start ');

compareArray(builder.\_stringArray, expectedArray);

});

it('should insertAt given index correctly', function(){

let expectedArray = Array.from('Param inserted eter');

builder.insertAt(' inserted ', 5);

compareArray(builder.\_stringArray, expectedArray);

});

it('should remove correctly', function(){

let expectedArray = Array.from('Parer');

builder.remove(3, 4);

compareArray(builder.\_stringArray, expectedArray);

});

it('should toString correctly', function(){

let expectedString = 'Parameter';

expect(builder.toString()).to.equal(expectedString);

});

it('should throw by invalid append parameter', function(){

let thrown = () => builder.append(8);

expect(thrown).to.throw();

});

it('should throw by invalid prepend parameter', function(){

let thrown = () => builder.prepend(8);

expect(thrown).to.throw();

});

it('should throw by invalid insertAt parameter', function(){

let thrown = () => builder.insertAt(8, 10);

expect(thrown).to.throw();

});

function compareArray(source, expected){

expect(source.length).to.equal(expected.length, 'Arrays do not match!');

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

expect(source[i]).to.equal(expected[i], `Element ${source[i]} at index ${i} mismatch with expected element ${expected[i]}!`);

}

}

});

});

|  |
| --- |
| class StringBuilder { |
|  | constructor(string) { |
|  | if (string !== undefined) { |
|  | StringBuilder.\_vrfyParam(string); |
|  | this.\_stringArray = Array.from(string); |
|  | } else { |
|  | this.\_stringArray = []; |
|  | } |
|  | } |
|  |  |
|  | append(string) { |
|  | StringBuilder.\_vrfyParam(string); |
|  | for (let i = 0; i < string.length; i++) { |
|  | this.\_stringArray.push(string[i]); |
|  | } |
|  | } |
|  |  |
|  | prepend(string) { |
|  | StringBuilder.\_vrfyParam(string); |
|  | for (let i = string.length - 1; i >= 0; i--) { |
|  | this.\_stringArray.unshift(string[i]); |
|  | } |
|  | } |
|  |  |
|  | insertAt(string, startIndex) { |
|  | StringBuilder.\_vrfyParam(string); |
|  | this.\_stringArray.splice(startIndex, 0, ...string); |
|  | } |
|  |  |
|  | remove(startIndex, length) { |
|  | this.\_stringArray.splice(startIndex, length); |
|  | } |
|  |  |
|  | static \_vrfyParam(param) { |
|  | if (typeof param !== 'string') throw new TypeError('Argument must be string'); |
|  | } |
|  |  |
|  | toString() { |
|  | return this.\_stringArray.join(''); |
|  | } |
|  | } |
|  |  |
|  | module.exports = StringBuilder; |

//let expect = require('chai').expect;

//let StringBuilder = require('./input');

describe('StringBuilder', function () {

let builder;

it('initialization does not throw', function () {

let initEmpty = () => builder = new StringBuilder();

expect(initEmpty).to.not.throw();

let initParam = () => builder = new StringBuilder('hello');

expect(initParam).to.not.throw();

});

it('invalid constructor parameter', function () {

let willThrow = () => builder = new StringBuilder(5);

expect(willThrow).to.throw();

});

describe('with empty constructor', function () {

beforeEach(function () {

builder = new StringBuilder();

});

it('has all properties', function () {

expect(builder.hasOwnProperty('\_stringArray')).to.equal(true, "Missing \_stringArray property");

});

it('has functions attached to prototype', function () {

expect(Object.getPrototypeOf(builder).hasOwnProperty('append')).to.equal(true, "Missing append function");

expect(Object.getPrototypeOf(builder).hasOwnProperty('prepend')).to.equal(true, "Missing prepend function");

expect(Object.getPrototypeOf(builder).hasOwnProperty('insertAt')).to.equal(true, "Missing insertAt function");

expect(Object.getPrototypeOf(builder).hasOwnProperty('remove')).to.equal(true, "Missing remove function");

expect(Object.getPrototypeOf(builder).hasOwnProperty('toString')).to.equal(true, "Missing toString function");

});

it('must initialize data to an empty array', function () {

expect(builder.\_stringArray instanceof Array).to.equal(true, 'Data must be of type array');

expect(builder.\_stringArray.length).to.equal(0, 'Data array must be initialized empty');

});

});

describe('constructor with parameter', function () {

let startingString = 'hello';

beforeEach(function () {

builder = new StringBuilder(startingString);

});

it('must initialize data to a string array', function () {

expect(builder.\_stringArray instanceof Array).to.equal(true, 'Data must be of type array');

compareArray(builder.\_stringArray, Array.from(startingString));

});

it('appends correctly', function () {

let str = ', world';

builder.append(str);

compareArray(builder.\_stringArray, Array.from(startingString + str));

});

it('prepends correctly', function () {

let str = 'welcome ';

builder.prepend(str);

compareArray(builder.\_stringArray, Array.from(str + startingString));

});

it('inserts correctly', function () {

let str = 'kek';

builder.insertAt(str, 3);

let expected = Array.from(startingString);

expected.splice(3, 0, ...str);

compareArray(builder.\_stringArray, expected);

});

it('removes correctly', function () {

builder.remove(1, 3);

compareArray(builder.\_stringArray, Array.from('ho'));

});

it('stringifies correctly', function () {

expect(builder.toString()).to.equal(startingString);

});

it('invalid append parameter', function () {

let willThrow = () => builder.append(5);

expect(willThrow).to.throw();

});

it('invalid prepend parameter', function () {

let willThrow = () => builder.prepend(5);

expect(willThrow).to.throw();

});

it('invalid insertAt parameter', function () {

let willThrow = () => builder.insertAt(5, 1);

expect(willThrow).to.throw();

});

});

function compareArray(source, expected) {

expect(source.length).to.equal(expected.length, "Arrays don't match");

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

expect(source[i]).to.equal(expected[i], 'Element ' + i + ' mismatch');

}

}

});

## 8.Payment Package

class PaymentPackage {

constructor(name, value) {

this.name = name;

this.value = value;

this.VAT = 20; // Default value

this.active = true; // Default value

}

get name() {

return this.\_name;

}

set name(newValue) {

if (typeof newValue !== 'string') {

throw new Error('Name must be a non-empty string');

}

if (newValue.length === 0) {

throw new Error('Name must be a non-empty string');

}

this.\_name = newValue;

}

get value() {

return this.\_value;

}

set value(newValue) {

if (typeof newValue !== 'number') {

throw new Error('Value must be a non-negative number');

}

if (newValue < 0) {

throw new Error('Value must be a non-negative number');

}

this.\_value = newValue;

}

get VAT() {

return this.\_VAT;

}

set VAT(newValue) {

if (typeof newValue !== 'number') {

throw new Error('VAT must be a non-negative number');

}

if (newValue < 0) {

throw new Error('VAT must be a non-negative number');

}

this.\_VAT = newValue;

}

get active() {

return this.\_active;

}

set active(newValue) {

if (typeof newValue !== 'boolean') {

throw new Error('Active status must be a boolean');

}

this.\_active = newValue;

}

toString() {

const output = [

`Package: ${this.name}` + (this.active === false ? ' (inactive)' : ''),

`- Value (excl. VAT): ${this.value}`,

`- Value (VAT ${this.VAT}%): ${this.value \* (1 + this.VAT / 100)}`

];

return output.join('\n');

}

}

module.exports = {PaymentPackage};

let expect = require('chai').expect;

let PaymentPackage = require('../08PaymentPackage').PaymentPackage;

//In Judge must be paste without this above

describe('PaymentPackage', function(){

describe('constructor with correct parameters', function(){

let paymentPackage;

beforeEach(function(){

paymentPackage = new PaymentPackage('testPaymentPackage', 60.5);

});

it('should create an instance with correct parameters', function(){

paymentPackage = new PaymentPackage('testPaymentPackage', 100);

expect(paymentPackage instanceof PaymentPackage).to.be.true;

});

it('should create an instance with correct property \_name', function(){

expect(paymentPackage.hasOwnProperty('\_name')).to.be.true;

});

it('should create an instance with correct property \_value', function(){

expect(paymentPackage.hasOwnProperty('\_value')).to.be.true;

});

it('should create an instance with correct property \_VAT', function(){

expect(paymentPackage.hasOwnProperty('\_VAT')).to.be.true;

});

it('should create an instance with correct property \_active', function(){

expect(paymentPackage.hasOwnProperty('\_active')).to.be.true;

});

it('should have correct value for name', function(){

expect(paymentPackage.name).to.be.equal('testPaymentPackage');

});

it('should have correct value for \_name', function(){

expect(paymentPackage.\_name).to.be.equal('testPaymentPackage');

});

it('should have correct value for value', function(){

expect(paymentPackage.value).to.be.closeTo(60.5, 0.01);

});

it('should have correct value for \_value', function(){

expect(paymentPackage.\_value).to.be.closeTo(60.5, 0.01);

});

it('should have correct value for VAT', function(){

expect(paymentPackage.VAT).to.be.closeTo(20, 0.01);

});

it('should have correct value for \_VAT', function(){

expect(paymentPackage.\_VAT).to.be.closeTo(20, 0.01);

});

it('should have correct value for active', function(){

expect(paymentPackage.active).to.be.true;

});

it('should have correct value for \_VAT', function(){

expect(paymentPackage.\_VAT).to.be.greaterThan(1);

});

it('should have correct value for \_active', function(){

expect(paymentPackage.\_active).to.be.true;

});

});

describe('constructor with incorrect parameters', function(){

let paymentPackage;

it('should throw for (120, 120)', function(){

expect(() => paymentPackage = new PaymentPackage(120, 120)).to.throw();

});

it('should throw for ("abc", "def")', function(){

expect(() => paymentPackage = new PaymentPackage("abc", "def")).to.throw();

});

it('should throw for ([], 120)', function(){

expect(() => paymentPackage = new PaymentPackage([], 120)).to.throw();

});

it('should throw for ({}, 120)', function(){

expect(() => paymentPackage = new PaymentPackage({}, 120)).to.throw();

});

it('should throw for (null, 120)', function(){

expect(() => paymentPackage = new PaymentPackage(null, 120)).to.throw();

});

it('should throw for (undefined, 120)', function(){

expect(() => paymentPackage = new PaymentPackage(undefined, 120)).to.throw();

});

it('should throw for ("", 120)', function(){

expect(() => paymentPackage = new PaymentPackage("", 120)).to.throw();

});

it('should throw for (new Date(2019, 10, 10), 120)', function(){

expect(() => paymentPackage = new PaymentPackage(new Date(2019, 10, 10), 120)).to.throw();

});

it('should throw for ("abc", [])', function(){

expect(() => paymentPackage = new PaymentPackage("abc", -1)).to.throw();

});

it('should throw for ("abc", [])', function(){

expect(() => paymentPackage = new PaymentPackage("abc", [])).to.throw();

});

it('should throw for ("abc", {})', function(){

expect(() => paymentPackage = new PaymentPackage("abc", {})).to.throw();

});

it('should throw for ("abc", null)', function(){

expect(() => paymentPackage = new PaymentPackage("abc", null)).to.throw();

});

it('should throw for ("abc", new Date(2019, 10, 10))', function(){

expect(() => paymentPackage = new PaymentPackage("abc", new Date(2019, 10, 10))).to.throw();

});

it('should throw for ("abc", undfined)', function(){

expect(() => paymentPackage = new PaymentPackage("abc", undefined)).to.throw();

});

it('should throw for ("abc", "")', function(){

expect(() => paymentPackage = new PaymentPackage("abc", "")).to.throw();

});

it('should throw for ("abc")', function(){

expect(() => paymentPackage = new PaymentPackage("abc")).to.throw();

});

});

describe("value - additon tests", function () {

it('should have correct value for value like 0.5', function(){

let paymentPackage = new PaymentPackage("abc", 0.5);

expect(paymentPackage.value).to.be.closeTo(0.5, 0.001);

});

});

describe('toString works correctly', function(){

it('should return correct value for ("abc", 100)', function(){

let paymentPackage = new PaymentPackage("abc", 100);

let expectedText = 'Package: abc\n- Value (excl. VAT): 100\n- Value (VAT 20%): 120';

let actualText = paymentPackage.toString();

expect(actualText).to.be.equal(expectedText);

});

it('should return correct value for ("abc", 100) and inactive', function(){

let paymentPackage = new PaymentPackage("abc", 100);

let expectedText = 'Package: abc (inactive)\n- Value (excl. VAT): 100\n- Value (VAT 20%): 120';

paymentPackage.active = false;

let actualText = paymentPackage.toString();

expect(actualText).to.be.equal(expectedText);

});

it('should return correct value for ("abc", 0) and inactive', function(){

let paymentPackage = new PaymentPackage("abc", 0);

let expectedText = 'Package: abc (inactive)\n- Value (excl. VAT): 0\n- Value (VAT 20%): 0';

paymentPackage.active = false;

let actualText = paymentPackage.toString();

expect(actualText).to.be.equal(expectedText);

});

});

// describe("toString", function () {

// it("should return correct value for ('asdasd', 500)", function () {

// let p = new PaymentPackage("asdasd", 500);

// let expectedText = [

// `Package: ${p.name}` + '',

// `- Value (excl. VAT): ${p.value}`,

// `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}`

// ].join("\n");

// let actualText = p.toString();

// expect(actualText).to.be.equal(expectedText);

// });

// // it("should return correct value for ('heyheyhey', 0.5)", function () {

// // let p = new PaymentPackage("heyheyhey", 0.5);

// // let expectedText = [

// // `Package: ${p.name}` + '',

// // `- Value (excl. VAT): ${p.value}`,

// // `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}`

// // ].join("\n");

// // let actualText = p.toString();

// // expect(actualText).to.be.equal(expectedText);

// // });

// // it("should return correct value for ('h', 5000)", function () {

// // let p = new PaymentPackage("h", 5000);

// // let expectedText = [

// // `Package: ${p.name}` + '',

// // `- Value (excl. VAT): ${p.value}`,

// // `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}`

// // ].join("\n");

// // let actualText = p.toString();

// // expect(actualText).to.be.equal(expectedText);

// // });

// // it("should return correct value for ('h', 5000) inactive", function () {

// // let p = new PaymentPackage("h", 5000);

// // p.active = false;

// // p.active = true;

// // p.active = false;

// // let expectedText = [

// // `Package: ${p.name}` + ' (inactive)',

// // `- Value (excl. VAT): ${p.value}`,

// // `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}`

// // ].join("\n");

// // let actualText = p.toString();

// // expect(actualText).to.be.equal(expectedText);

// // });

// it("should return correct value for ('123123123', 123) inactive", function () {

// let p = new PaymentPackage("123123123", 123);

// p.active = false;

// p.active = true;

// p.active = false;

// let expectedText = [

// `Package: ${p.name}` + ' (inactive)',

// `- Value (excl. VAT): ${p.value}`,

// `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}`

// ].join("\n");

// let actualText = p.toString();

// expect(actualText).to.be.equal(expectedText);

// });

// // it("should return correct value for ('123123123', 0.9) inactive", function () {

// // let p = new PaymentPackage("123123123", 0.9);

// // p.active = false;

// // p.active = true;

// // p.active = false;

// // let expectedText = [

// // `Package: ${p.name}` + ' (inactive)',

// // `- Value (excl. VAT): ${p.value}`,

// // `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}`

// // ].join("\n");

// // let actualText = p.toString();

// // expect(actualText).to.be.equal(expectedText);

// // });

// it("should return correct value for ('0', 0) inactive", function () {

// let p = new PaymentPackage("0", 0);

// p.active = false;

// p.active = true;

// p.active = false;

// let expectedText = [

// `Package: ${p.name}` + ' (inactive)',

// `- Value (excl. VAT): ${p.value}`,

// `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}`

// ].join("\n");

// let actualText = p.toString();

// expect(actualText).to.be.equal(expectedText);

// });

// })

});

|  |
| --- |
| class PaymentPackage { |
|  | constructor(name, value) { |
|  | this.name = name; |
|  | this.value = value; |
|  | this.VAT = 20; // Default value |
|  | this.active = true; // Default value |
|  | } |
|  |  |
|  | get name() { |
|  | return this.\_name; |
|  | } |
|  |  |
|  | set name(newValue) { |
|  | if (typeof newValue !== 'string') { |
|  | throw new Error('Name must be a non-empty string'); |
|  | } |
|  | if (newValue.length === 0) { |
|  | throw new Error('Name must be a non-empty string'); |
|  | } |
|  | this.\_name = newValue; |
|  | } |
|  |  |
|  | get value() { |
|  | return this.\_value; |
|  | } |
|  |  |
|  | set value(newValue) { |
|  | if (typeof newValue !== 'number') { |
|  | throw new Error('Value must be a non-negative number'); |
|  | } |
|  | if (newValue < 0) { |
|  | throw new Error('Value must be a non-negative number'); |
|  | } |
|  | this.\_value = newValue; |
|  | } |
|  |  |
|  | get VAT() { |
|  | return this.\_VAT; |
|  | } |
|  |  |
|  | set VAT(newValue) { |
|  | if (typeof newValue !== 'number') { |
|  | throw new Error('VAT must be a non-negative number'); |
|  | } |
|  | if (newValue < 0) { |
|  | throw new Error('VAT must be a non-negative number'); |
|  | } |
|  | this.\_VAT = newValue; |
|  | } |
|  |  |
|  | get active() { |
|  | return this.\_active; |
|  | } |
|  |  |
|  | set active(newValue) { |
|  | if (typeof newValue !== 'boolean') { |
|  | throw new Error('Active status must be a boolean'); |
|  | } |
|  | this.\_active = newValue; |
|  | } |
|  |  |
|  | toString() { |
|  | const output = [ |
|  | `Package: ${this.name}` + (this.active === false ? ' (inactive)' : ''), |
|  | `- Value (excl. VAT): ${this.value}`, |
|  | `- Value (VAT ${this.VAT}%): ${this.value \* (1 + this.VAT / 100)}` |
|  | ]; |
|  | return output.join('\n'); |
|  | } |
|  | } |
|  |  |
|  | module.exports = PaymentPackage; |

|  |
| --- |
| //one zero test not past in Judge  const expect = require("chai").expect; |
|  | const PaymentPackage = require("../2\_PaymentPackage"); |
|  |  |
|  |  |
|  | describe("PaymentPackage", function () { |
|  | describe("constructor", function () { |
|  | it("should create an instance for ('testP, 100')", function name() { |
|  | let p = new PaymentPackage("testP", 100); |
|  |  |
|  | expect(p instanceof PaymentPackage).to.be.true; |
|  | }); |
|  | it("should create an instance with correct properties for ('testP, 50')", function name() { |
|  | let p = new PaymentPackage("testP", 50); |
|  |  |
|  | expect(p.hasOwnProperty("\_name")).to.be.true; |
|  | expect(p.hasOwnProperty("\_value")).to.be.true; |
|  | expect(p.hasOwnProperty("\_VAT")).to.be.true; |
|  | expect(p.hasOwnProperty("\_active")).to.be.true; |
|  | }); |
|  | it("should have correct values for ('test3', 60.5)", function name() { |
|  | let p = new PaymentPackage('test3', 60.5); |
|  |  |
|  | expect(p.name).to.be.equal("test3"); |
|  | expect(p.value).to.be.closeTo(60.5, 0.01); |
|  | expect(p.VAT).to.be.closeTo(20, 0.01); |
|  | expect(p.active).to.be.true; |
|  | expect(p.\_active).to.be.true; |
|  | expect(p.VAT).to.be.greaterThan(1); |
|  | expect(p.\_VAT).to.be.greaterThan(1); |
|  | expect(p.value).to.be.greaterThan(50); |
|  | expect(p.\_value).to.be.greaterThan(50); |
|  | }); |
|  | it("should throw error when initialised with incorrect values", function () { |
|  | let p = null; |
|  | expect(() => p = new PaymentPackage(123, 123)).to.throw(Error); |
|  | expect(() => p = new PaymentPackage("correct", "asd")).to.throw(Error); |
|  | expect(() => p = new PaymentPackage([], 10.50)).to.throw(Error); |
|  | expect(() => p = new PaymentPackage(null, 60.50)).to.throw(Error); |
|  | expect(() => p = new PaymentPackage("ttt", {})).to.throw(Error); |
|  | expect(() => p = new PaymentPackage("tttt", new Date(2010, 10, 10))).to.throw(Error); |
|  | expect(() => p = new PaymentPackage(null, null)).to.throw(Error); |
|  | expect(() => p = new PaymentPackage(undefined, undefined)).to.throw(Error); |
|  | expect(() => p = new PaymentPackage("ok")).to.throw(Error); |
|  | expect(() => p = new PaymentPackage("", 123)).to.throw(Error); |
|  | expect(() => p = new PaymentPackage("asd", -1)).to.throw(Error); |
|  | expect(() => p = new PaymentPackage("asd", -50)).to.throw(Error); |
|  | expect(() => p = new PaymentPackage("", -2)).to.throw(Error); |
|  | }) |
|  | }); |
|  |  |
|  | describe("name", function () { |
|  | it("should create instance with corrrect name for ('abc', 10)", function () { |
|  | let p = new PaymentPackage("abc", 10); |
|  |  |
|  | expect(p.name).to.be.equal("abc"); |
|  | expect(p.\_name).to.be.equal("abc"); |
|  | expect(p.hasOwnProperty("\_name")).to.be.true; |
|  | }); |
|  | it("should create instance with corrrect name for ('dasdasdasdasdasdas', 10)", function () { |
|  | let p = new PaymentPackage("dasdasdasdasdasdas", 10); |
|  |  |
|  | expect(p.name).to.be.equal("dasdasdasdasdasdas"); |
|  | expect(p.\_name).to.be.equal("dasdasdasdasdasdas"); |
|  | expect(p.hasOwnProperty("\_name")).to.be.true; |
|  | }); |
|  | it("should create instance with corrrect name for ('a', 10)", function () { |
|  | let p = new PaymentPackage("a", 10); |
|  |  |
|  | expect(p.name).to.be.equal("a"); |
|  | expect(p.\_name).to.be.equal("a"); |
|  | expect(p.hasOwnProperty("\_name")).to.be.true; |
|  | }); |
|  | }); |
|  |  |
|  | describe("value", function () { |
|  | it("should create instance with corrrect name for ('abc', 10)", function () { |
|  | let p = new PaymentPackage("abc", 10); |
|  |  |
|  | expect(p.value).to.be.closeTo(10, 0.01); |
|  | expect(p.\_value).to.be.closeTo(10, 0.01); |
|  | expect(p.hasOwnProperty("\_value")).to.be.true; |
|  | }); |
|  | it("should create instance with corrrect name for ('abc', 0.5)", function () { |
|  | let p = new PaymentPackage("abc", 0.5); |
|  |  |
|  | expect(p.value).to.be.closeTo(0.5, 0.001); |
|  | expect(p.\_value).to.be.closeTo(0.5, 0.001); |
|  | expect(p.hasOwnProperty("\_value")).to.be.true; |
|  | }); |
|  | it("should throw error for ('abc', -10)", function () { |
|  | let p = undefined; |
|  |  |
|  | expect(() => p = new PaymentPackage("abc", -10)).to.throw(Error); |
|  | }); |
|  | }); |
|  |  |
|  | describe("VAT", function () { |
|  | let p = null; |
|  | beforeEach(function () { |
|  | let p = new PaymentPackage("abc", 10); |
|  | }); |
|  | it("should have default property VAT with default value", function () { |
|  | expect(p.VAT).to.be.closeTo(20, 0.01); |
|  | expect(p.\_VAT).to.be.closeTo(20, 0.01); |
|  | expect(p.hasOwnProperty("\_VAT")).to.be.true; |
|  | }); |
|  | it("should change value for correct input for VAT 30", function () { |
|  | p.VAT = 30; |
|  | expect(p.VAT).to.be.closeTo(30, 0.01); |
|  | }); |
|  | it("should change value for correct input for VAT 10.5", function () { |
|  | p.VAT = 10.5; |
|  | expect(p.VAT).to.be.closeTo(10.5, 0.001); |
|  | }); |
|  | it("should change value for correct input for VAT 0", function () { |
|  | p.VAT = 0; |
|  | expect(p.VAT).to.be.closeTo(0, 0.001); |
|  | }); |
|  | it("should throw error for invalid values for VAT", function () { |
|  | expect(() => p.VAT = -1).to.throw(Error); |
|  | expect(() => p.VAT = -0.1).to.throw(Error); |
|  | expect(() => p.VAT = -0.0001).to.throw(Error); |
|  | expect(() => p.VAT = []).to.throw(Error); |
|  | expect(() => p.VAT = {}).to.throw(Error); |
|  | expect(() => p.VAT = "asd").to.throw(Error); |
|  | expect(() => p.VAT = null).to.throw(Error); |
|  | expect(() => p.VAT = undefined).to.throw(Error); |
|  | }) |
|  | }); |
|  |  |
|  | describe("active", function () { |
|  | let p = null; |
|  | beforeEach(function () { |
|  | let p = new PaymentPackage("abc", 10); |
|  | }); |
|  |  |
|  | it("should have default value when instantiated", function () { |
|  | expect(p.active).to.be.true; |
|  | expect(p.\_active).to.be.true; |
|  | expect(p.hasOwnProperty("\_active")).to.be.true; |
|  | }); |
|  |  |
|  | it("should change value for correct input", function () { |
|  | p.active = false; |
|  | expect(p.active).to.be.false; |
|  | expect(p.\_active).to.be.false; |
|  | p.active = true; |
|  | expect(p.active).to.be.true; |
|  | expect(p.\_active).to.be.true; |
|  | p.active = false; |
|  | p.active = true; |
|  | p.active = false; |
|  | expect(p.active).to.be.false; |
|  | expect(p.\_active).to.be.false; |
|  | }); |
|  | it("should throw error for invalid values for active", function () { |
|  | expect(() => p.active = -11).to.throw(Error); |
|  | expect(() => p.active = -0.1).to.throw(Error); |
|  | expect(() => p.active = -0.0001).to.throw(Error); |
|  | expect(() => p.active = []).to.throw(Error); |
|  | expect(() => p.active = {}).to.throw(Error); |
|  | expect(() => p.active = "asd").to.throw(Error); |
|  | expect(() => p.active = null).to.throw(Error); |
|  | expect(() => p.active = undefined).to.throw(Error); |
|  | }); |
|  | }); |
|  |  |
|  | describe("toString", function () { |
|  | it("should return correct value for ('asdasd', 500)", function () { |
|  | let p = new PaymentPackage("asdasd", 500); |
|  | let expectedText = [ |
|  | `Package: ${p.name}` + '', |
|  | `- Value (excl. VAT): ${p.value}`, |
|  | `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}` |
|  | ].join("\n"); |
|  | let actualText = p.toString(); |
|  |  |
|  | expect(actualText).to.be.equal(expectedText); |
|  | }); |
|  | it("should return correct value for ('heyheyhey', 0.5)", function () { |
|  | let p = new PaymentPackage("heyheyhey", 0.5); |
|  | let expectedText = [ |
|  | `Package: ${p.name}` + '', |
|  | `- Value (excl. VAT): ${p.value}`, |
|  | `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}` |
|  | ].join("\n"); |
|  | let actualText = p.toString(); |
|  |  |
|  | expect(actualText).to.be.equal(expectedText); |
|  | }); |
|  | it("should return correct value for ('h', 5000)", function () { |
|  | let p = new PaymentPackage("h", 5000); |
|  | let expectedText = [ |
|  | `Package: ${p.name}` + '', |
|  | `- Value (excl. VAT): ${p.value}`, |
|  | `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}` |
|  | ].join("\n"); |
|  | let actualText = p.toString(); |
|  |  |
|  | expect(actualText).to.be.equal(expectedText); |
|  | }); |
|  |  |
|  | it("should return correct value for ('h', 5000) inactive", function () { |
|  | let p = new PaymentPackage("h", 5000); |
|  | p.active = false; |
|  | p.active = true; |
|  | p.active = false; |
|  | let expectedText = [ |
|  | `Package: ${p.name}` + ' (inactive)', |
|  | `- Value (excl. VAT): ${p.value}`, |
|  | `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}` |
|  | ].join("\n"); |
|  | let actualText = p.toString(); |
|  |  |
|  | expect(actualText).to.be.equal(expectedText); |
|  | }); |
|  | it("should return correct value for ('123123123', 123) inactive", function () { |
|  | let p = new PaymentPackage("123123123", 123); |
|  | p.active = false; |
|  | p.active = true; |
|  | p.active = false; |
|  | let expectedText = [ |
|  | `Package: ${p.name}` + ' (inactive)', |
|  | `- Value (excl. VAT): ${p.value}`, |
|  | `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}` |
|  | ].join("\n"); |
|  | let actualText = p.toString(); |
|  |  |
|  | expect(actualText).to.be.equal(expectedText); |
|  | }); |
|  | it("should return correct value for ('123123123', 0.9) inactive", function () { |
|  | let p = new PaymentPackage("123123123", 0.9); |
|  | p.active = false; |
|  | p.active = true; |
|  | p.active = false; |
|  | let expectedText = [ |
|  | `Package: ${p.name}` + ' (inactive)', |
|  | `- Value (excl. VAT): ${p.value}`, |
|  | `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}` |
|  | ].join("\n"); |
|  | let actualText = p.toString(); |
|  |  |
|  | expect(actualText).to.be.equal(expectedText); |
|  | }); |
|  | it("should return correct value for ('0', 0) inactive", function () { |
|  | let p = new PaymentPackage("0", 0); |
|  | p.active = false; |
|  | p.active = true; |
|  | p.active = false; |
|  | let expectedText = [ |
|  | `Package: ${p.name}` + ' (inactive)', |
|  | `- Value (excl. VAT): ${p.value}`, |
|  | `- Value (VAT ${p.VAT}%): ${p.value \* (1 + p.VAT / 100)}` |
|  | ].join("\n"); |
|  | let actualText = p.toString(); |
|  |  |
|  | expect(actualText).to.be.equal(expectedText); |
|  | }); |
|  | }) |
|  | }); |