# Solutions-Exercises: Classes

## 1.Data Class

class HTTPRequestData{

constructor(method, uri, version, message){

this.method = method;

this.uri = uri;

this.version = version;

this.message = message;

this.response = undefined;

this.fulfilled = false;

}

}

let httpRequestData = new HTTPRequestData('GET', 'http://google.com', 'HTTP/1.1', '')

console.log(httpRequestData)

|  |
| --- |
| class Data { |
|  | constructor(method, uri, version, message) { |
|  | this.method = method; |
|  | this.uri = uri; |
|  | this.version = version; |
|  | this.message = message; |
|  | this.response = undefined; |
|  | this.fulfilled = false; |
|  | } |
|  | } |

## 2.Tickets

function sortTickets(ticketsData, sortingCriterion) {

class Ticket {

constructor(destination, price, status) {

this.destination = destination;

this.price = price;

this.status = status;

}

}

let tickets = [];

for (let ticketData of ticketsData) {

let [destination, price, status] = ticketData.split('|');

price = Number(price);

let ticket = new Ticket(destination, price, status);

tickets.push(ticket);

}

tickets = tickets.filter(t => t !== '');

switch(sortingCriterion){

case 'destination': tickets.sort((a, b) => a.destination.localeCompare(b.destination)); break;

case 'price': tickets.sort((a, b) => a.price - b.price); break;

case 'status': tickets.sort((a, b) => a.status.localeCompare(b.status)); break;

}

return tickets;

}

console.log(sortTickets(['Philadelphia|94.20|available', 'New York City|95.99|available', 'New York City|95.99|sold', 'Boston|126.20|departed'], 'destination'))

console.log(sortTickets(['Philadelphia|94.20|available', 'New York City|95.99|available', 'New York City|95.99|sold', 'Boston|126.20|departed'], 'status'))

|  |
| --- |
| function solve(arr, str) { |
|  | class Ticket { |
|  | constructor(destination, price, status) { |
|  | this.destination = destination; |
|  | this.price = price; |
|  | this.status = status; |
|  | } |
|  | } |
|  |  |
|  | let result = []; |
|  | arr.forEach(description => { |
|  | let [destination, price, status] = description.split(/**\|**/g).filter(w => w !== ""); |
|  | price = Number(price); |
|  | result.push(new Ticket(destination, price, status)); |
|  | }); |
|  |  |
|  | return result.sort(function (a, b) { |
|  | if (str === "price") { |
|  | return a[str] - b[str]; |
|  | } else { |
|  | return a[str].localeCompare(b[str]); |
|  | } |
|  | }); |
|  | } |
|  |  |
|  | console.log(solve(['Philadelphia|94.20|available', |
|  | 'New York City|95.99|available', |
|  | 'New York City|95.99|sold', |
|  | 'Boston|126.20|available', |
|  | 'Philadelphia|132.20|departed', |
|  | 'Chicago|140.20|available', |
|  | 'Dallas|144.60|sold', |
|  | 'New York City|206.20|sold', |
|  | 'New York City|240.20|departed', |
|  | 'New York City|305.20|departed' |
|  | ], |
|  | 'destination')); |

## 3.Unity

class Rat{

constructor(name){

this.name = name;

this.unitedRats = [];

}

unite(rat){

if(rat instanceof Rat){

this.unitedRats.push(rat);

}

}

getRats(){

return this.unitedRats;

}

toString(){

let ratNames = this.unitedRats.map(r => `##${r.name}`)

return this.unitedRats.length === 0 ? this.name : this.name + '\n' + ratNames.join('\n');

}

}

let rat = new Rat("Pesho");

console.log(rat.toString()); //Pesho

console.log(rat.getRats()); //[]

rat.unite(new Rat("Gosho"));

rat.unite(new Rat("Sasho"));

console.log(rat.getRats());

//[ Rat { name: 'Gosho', unitedRats: [] },

// Rat { name: 'Sasho', unitedRats: [] } ]

console.log(rat.toString());

// Pesho

// ##Gosho

// ##Sasho

|  |
| --- |
| class Rat { |
|  | constructor(name) { |
|  | this.name = name; |
|  | this.otherRats = []; |
|  | } |
|  |  |
|  | unite(otherRat) { |
|  | if (otherRat instanceof Rat) { |
|  | this.otherRats.push(otherRat); |
|  | } |
|  | } |
|  |  |
|  | getRats() { |
|  | return this.otherRats; |
|  | } |
|  |  |
|  | toString() { |
|  | let result = this.name; |
|  | if (this.otherRats.length > 0) { |
|  | result += "\n"; |
|  | } |
|  | this.otherRats.forEach(r => { |
|  | result += `##${r.name}\n`; |
|  | }); |
|  |  |
|  | return result; |
|  | } |
|  | } |
|  |  |
|  | let test = new Rat("Pesho"); |
|  |  |
|  | test.unite(new Rat("Gosho")); |
|  | test.unite(new Rat("Sasho")); |
|  |  |
|  | console.log(test.toString()); |

## 4.Length Limit

class Stringer{

constructor(innerString, innerLength){

this.innerString = innerString;

this.\_innerLength = innerLength;

}

get innerLength(){return this.\_innerLength;}

set innerLength(innerLength){

//innerLength = Number(innerLength);

if (innerLength < 0){

innerLength = 0;

}

this.\_innerLength = innerLength;

}

increase(length){

this.innerLength += length;

}

decrease(length){

this.innerLength -= length;

}

toString(){

let stringToReturn = this.innerString.substr(0, this.innerLength);

if (this.innerString.length > this.innerLength){

stringToReturn += '...';

}

return stringToReturn;

}

}

let str = new Stringer("Test", 5);

console.log(str.toString()); //Test

str.decrease(3);

console.log(str.toString()); //Te...

str.decrease(5);

console.log(str.toString()); //...

str.increase(4);

console.log(str.toString()); //Test

|  |
| --- |
| class Stringer { |
|  | constructor(str, len) { |
|  | this.innerString = str; |
|  | this.innerLength = len; |
|  | } |
|  |  |
|  | increase(length) { |
|  | if (Number.isInteger(length)) { |
|  | this.innerLength += length; |
|  | } |
|  | } |
|  | decrease(length) { |
|  | if (Number.isInteger(length)) { |
|  | this.innerLength -= length; |
|  | this.innerLength = this.innerLength < 0 ? 0 : this.innerLength; |
|  | } |
|  | } |
|  |  |
|  | toString() { |
|  | if (this.innerLength === 0) { |
|  | return "..."; |
|  | } |
|  | if (this.innerString.length > this.innerLength) { |
|  | return this.innerString.substr(0, this.innerLength) + "..."; |
|  | } |
|  | return this.innerString; |
|  | } |
|  | } |
|  |  |
|  | let test = new Stringer("Test", 5); |
|  | console.log(test.toString()); //Test |
|  |  |
|  | test.decrease(3); |
|  | console.log(test.toString()); //Te... |
|  |  |
|  | test.decrease(5); |
|  | console.log(test.toString()); //... |
|  |  |
|  | test.increase(4); |
|  | console.log(test.toString()); //Test |

|  |
| --- |
| class Stringer{ |
|  | constructor(string, length){ |
|  | this.innerString = string; |
|  | this.innerLength = length; |
|  | } |
|  |  |
|  | increase(length){ |
|  | this.innerLength += length; |
|  | if(this.innerLength < 0){ |
|  | this.innerLength = 0; |
|  | } |
|  | } |
|  |  |
|  | decrease(length){ |
|  | this.innerLength -= length; |
|  | if(this.innerLength < 0){ |
|  | this.innerLength = 0; |
|  | } |
|  | } |
|  |  |
|  | toString(){ |
|  | if(this.innerLength == 0){ |
|  | return "..."; |
|  | } |
|  |  |
|  | if(this.innerString.length > this.innerLength){ |
|  | this.innerString = this.innerString.substr(0, this.innerLength) + "..."; |
|  | } |
|  | return this.innerString; |
|  | } |
|  | } |

## 5.\*Extensible Class

let extensible = //in Judge must be paste only the IIFE

(function extendClass(){

let counter = 0;

class Extensible{

constructor(){

this.id = counter;

counter++;

}

extend(template){

for(let property in template){

if(typeof template[property] === 'function'){

Extensible.prototype[property] = template[property];

//this.prototype[property] = template[property];

//Object.getPrototypeOf(this)[property] = template[property];

} else {

this[property] = template[property];

//Extensible[property] = template[property];

}

}

}

}

return Extensible;

})()

let obj1 = new extensible();

let obj2 = new extensible();

let obj3 = new extensible();

console.log(obj1.id);

console.log(obj2.id);

console.log(obj3.id);

|  |
| --- |
| (function solve() { |
|  | let id = 0; |
|  | class Extensible { |
|  | constructor() { |
|  | this.id = id++; |
|  | } |
|  | extend(template) { |
|  | for (let prop in template) { |
|  | if (typeof template[prop] == 'function') |
|  | Extensible.prototype[prop] = template[prop]; |
|  | else |
|  | this[prop] = template[prop]; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | return Extensible; |
|  | })() |

|  |
| --- |
| (() => { |
|  | let counter = 0; |
|  | return class Extensible{ |
|  | constructor(){ |
|  | this.id = counter; |
|  | counter++; |
|  | } |
|  | extend(template){ |
|  | for(let parentProp of Object.keys(template)){ |
|  | if(typeof(template[parentProp]) == "function"){ |
|  | Object.getPrototypeOf(this)[parentProp] = template[parentProp]; |
|  | } else { |
|  | this[parentProp] = template[parentProp]; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  | )() |

## 6.Sorted List

class SortedList{

constructor(){

this.sortedList = [];

this.size = 0;

}

add(element){

this.sortedList.push(element);

this.sortedList.sort((a, b) => a - b);

this.size++;

}

remove(index){

if(index < 0 || index >= this.size){

//return;

throw new Error('Incorrect index!');

}

this.sortedList.splice(index, 1);

this.size--;

}

get(index){

if(index < 0 || index >= this.size){

//return;

throw new Error('Incorrect index!');

}

return this.sortedList[index];

}

}

|  |
| --- |
| class Collection { |
|  | constructor() { |
|  | this.collection = []; |
|  | this.size = 0; |
|  | } |
|  |  |
|  | add(ele) { |
|  | this.collection.push(ele); |
|  | this.size++; |
|  | this.collection = this.collection.sort((a, b) => a - b); |
|  | } |
|  |  |
|  | remove(ind) { |
|  | this.collection[ind] = undefined; |
|  | this.collection = this.collection.filter(i => i !== undefined).sort((a, b) => a - b); |
|  | this.size = this.collection.length; |
|  | } |
|  |  |
|  | get(ind) { |
|  | return this.collection[ind]; |
|  | } |
|  | } |

|  |
| --- |
|  |
| class SortedList{ |
|  | constructor(){ |
|  | this.arr = []; |
|  | this.size = 0; |
|  | } |
|  |  |
|  | add(element){ |
|  | this.arr.push(element); |
|  | this.arr.sort((a,b) => a-b); |
|  | this.size++; |
|  | return this.arr; |
|  | }; |
|  |  |
|  | remove(index){ |
|  | if(index >=0 && index< this.arr.length) { |
|  | this.arr.splice(index, 1); |
|  | this.arr.sort((a,b) => a-b); |
|  | this.size--; |
|  | return this.arr; |
|  | } |
|  | } |
|  |  |
|  | get(index){ |
|  | if(index >= 0 && index < this.arr.length){ |
|  | return this.arr[index]; |
|  | } |
|  | } |
|  | } |
|  |  |

## 7.Instance Validation

class CheckingAccount{

constructor(clientId, email, firstName, lastName){

this.clientId = clientId;

this.email = email;

this.firstName = firstName;

this.lastName = lastName;

}

get clientId(){return this.\_clientId;}

set clientId(clientId){

let pattern = /^\d{6}$/;

if (!pattern.test(clientId)){

throw new TypeError('Client ID must be a 6-digit number');

}

this.\_clientId = clientId;

console.log(this.\_clientId)

//this.\_clientId = Number(clientId);

}

get email(){return this.\_email;}

set email(email){

let pattern = /^[A-Za-z0-9]+@[A-Za-z\_.]+$/;

if(!pattern.test(email)){

throw new TypeError('Invalid e-mail');

}

this.\_email = email;

}

get firstName(){return this.\_firstName;}

set firstName(firstName){

if(firstName.length <= 3 || firstName.length >= 20){

throw new TypeError('First name must be between 3 and 20 characters long');

}

let pattern = /^[A-Za-z]+$/;

if(!pattern.test(firstName)){

throw new TypeError('First name must contain only Latin characters');

}

this.\_firstName = firstName;

}

get lastName(){return this.\_lastName;}

set lastName(lastName){

if(lastName.length <= 3 || lastName.length >= 20){

throw new TypeError('Last name must be between 3 and 20 characters long');

}

let pattern = /^[A-Za-z]+$/;

if(!pattern.test(lastName)){

throw new TypeError('Last name must contain only Latin characters');

}

this.\_lastName = lastName;

}

}

let acc = new CheckingAccount('1314', 'ivan@some.com', 'Ivan', 'Petrov')

//let acc = new CheckingAccount('131455', 'ivan@', 'Ivan', 'Petrov')

//let acc = new CheckingAccount('131455', 'ivan@some.com', 'I', 'Petrov')

// let acc = new CheckingAccount('131455', 'ivan@some.com', 'Ivan', 'P3trov')

|  |
| --- |
| class CheckingAccount { |
|  | constructor(clientId, email, firstName, lastName) { |
|  | this.clientId = clientId; |
|  | this.email = email; |
|  | this.firstName = firstName; |
|  | this.lastName = lastName; |
|  | } |
|  |  |
|  | get clientId() { |
|  | return this.\_clientId; |
|  | } |
|  | set clientId(value) { |
|  | let regex = /^\d{6}$/g; |
|  | if (regex.test(value)) { |
|  | this.\_clientId = Number(value); |
|  | } else { |
|  | throw new TypeError("Client ID must be a 6-digit number"); |
|  | } |
|  | } |
|  |  |
|  | get email() { |
|  | return this.\_email; |
|  | } |
|  | set email(value) { |
|  | let regex = /[a-zA-Z0-9]@[a-zA-Z**\.**]+/g; |
|  | if (regex.test(value)) { |
|  | this.\_email = value; |
|  | } else { |
|  | throw new TypeError("Invalid e-mail"); |
|  | } |
|  | } |
|  |  |
|  | get firstName() { |
|  | return this.\_firstName; |
|  | } |
|  |  |
|  | set firstName(value) { |
|  | let regex1 = /^[a-zA-Z]+$/g; |
|  | let regex2 = /^.{3,20}$/g; |
|  | if (regex1.test(value) && regex2.test(value)) { |
|  | this.\_firstName = value; |
|  | } else { |
|  | if (!regex2.test(value)) { |
|  | throw new TypeError("First name must be between 3 and 20 characters long"); |
|  | } else { |
|  | throw new TypeError("First name must contain only Latin characters"); |
|  | } |
|  |  |
|  | } |
|  | } |
|  |  |
|  | get lastName() { |
|  | return this.\_lastName; |
|  | } |
|  |  |
|  | set lastName(value) { |
|  | let regex1 = /^[a-zA-Z]+$/g; |
|  | let regex2 = /^.{3,20}$/g; |
|  | if (regex1.test(value) && regex2.test(value)) { |
|  | this.\_lastName = value; |
|  | } else { |
|  | if (!regex2.test(value)) { |
|  | throw new TypeError("Last name must be between 3 and 20 characters long"); |
|  | } else { |
|  | throw new TypeError("Last name must contain only Latin characters"); |
|  | } |
|  |  |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // let acc = new CheckingAccount('131455', 'ivan@some.com', 'I', 'Petrov') |

# JS Advanced - Retake Exam: 18.11.2018

## 8. \*\*Kitchen

class Kitchen{

constructor(budget){

this.budget = budget;

this.menu = {};

this.productsInStock = {};

this.actionsHistory = [];

}

loadProducts(products){

for(let product of products){

let [productName, productQuantity, productPrice] = product.split(' ');

productQuantity = Number(productQuantity);

productPrice = Number(productPrice);

if(this.budget >= productPrice){

if(!this.productsInStock.hasOwnProperty(productName)){

this.productsInStock[productName] = 0;

}

this.productsInStock[productName] += productQuantity;

this.budget -= productPrice;

this.actionsHistory.push(`Successfully loaded ${productQuantity} ${productName}`);

}

else {

this.actionsHistory.push(`There was not enough money to load ${productQuantity} ${productName}`);

}

}

//return this.actionsHistory.join('\n').trim() + '\n';

//return this.actionsHistory.join('\n') + '\n';

return this.actionsHistory.join('\n');

}

addToMenu(meal, neededProducts, price){

if(!this.menu.hasOwnProperty(meal)){

this.menu[meal] = {meal, products: neededProducts, price};

let mealsCount = Object.keys(this.menu).length;

return `Great idea! Now with the ${meal} we have ${mealsCount} meals in the menu, other ideas?`;

}

else {

return `The ${meal} is already in our menu, try something different.`;

}

}

showTheMenu(){

if(Object.keys(this.menu).length > 0){

let menuList = Object.values(this.menu).map(m => `${m.meal} - $ ${m.price}`);

//return menuList.join('\n').trim() + '\n';

return menuList.join('\n') + '\n';

}

else {

return 'Our menu is not ready yet, please come later...';

}

}

makeTheOrder(meal){

if(!this.menu.hasOwnProperty(meal)){

return `There is not ${meal} yet in our menu, do you want to order something else?`

}

let neededProducts = this.menu[meal].products;

for(let neededProduct of neededProducts){

let [productName, productQuantity] = neededProduct.split(/\s+/);

productQuantity = Number(productQuantity);

if(!this.productsInStock.hasOwnProperty(productName) || this.productsInStock[productName] < productQuantity){

return `For the time being, we cannot complete your order (${meal}), we are very sorry...`;

}

}

neededProducts.forEach(neededProduct => {

let [productName, productQuantity] = neededProduct.split(/\s+/);

productQuantity = Number(productQuantity);

this.productsInStock[productName] -= productQuantity;

});

let price = this.menu[meal].price;

this.budget += price;

return `Your order (${meal}) will be completed in the next 30 minutes and will cost you ${price}.`

}

}

let kitchen = new Kitchen (1000);

console.log(kitchen.loadProducts(['Banana 10 5', 'Banana 20 10', 'Strawberries 50 30', 'Yogurt 10 10', 'Yogurt 500 1500', 'Honey 5 50']));

console.log(kitchen.addToMenu('frozenYogurt', ['Yogurt 1', 'Honey 1', 'Banana 1', 'Strawberries 10'], 9.99));

console.log(kitchen.addToMenu('Pizza', ['Flour 0.5', 'Oil 0.2', 'Yeast 0.5', 'Salt 0.1', 'Sugar 0.1', 'Tomato sauce 0.5', 'Pepperoni 1', 'Cheese 1.5'], 15.55));

console.log(kitchen.showTheMenu());

class Kitchen{

constructor(budget){

this.budget = budget;

this.menu = {};

this.productsInStock = {};

this.actionsHistory = [];

}

loadProducts(arr){

for (const line of arr) {

let productLine=line.split(' ');

let name=productLine[0];

let quantity=Number(productLine[1]);

let price=Number(productLine[2]);

let successString=`Successfully loaded ${quantity} ${name}`;

let failureString=`There was not enough money to load ${quantity} ${name}`;

let canBuy=this.budget-price>=0;

if(!this.productsInStock.hasOwnProperty(name) && canBuy){

this.productsInStock[name]=quantity;

this.budget-=price

this.actionsHistory.push(successString);

}else if(canBuy){

this.productsInStock[name] += quantity;

this.budget-=price;

this.actionsHistory.push(successString);

}else{

this.actionsHistory.push(failureString);

}

}

return this.actionsHistory.join('\n').trim()+'\n';

}

addToMenu(meal,products,price){

let exist=this.menu.hasOwnProperty(meal);

if(exist){

return `The ${meal} is already in our menu, try something different.`;

}

this.menu[meal] = {meal, products: products, price};

let count=Object.keys(this.menu).length;

return `Great idea! Now with the ${meal} we have ${count} meals in the menu, other ideas?`;

}

showTheMenu(){

if(Object.keys(this.menu).length===0){

return "Our menu is not ready yet, please come later...";

}

let result=Object.values(this.menu).map(x=>(`${x.meal} - $ ${x.price}`));

return result.join('\n').trim()+'\n';

}

makeTheOrder(meal) {

if (!this.menu.hasOwnProperty(meal)) {

return `There is not ${meal} yet in our menu, do you want to order something else?`

}

let neededProducts = this.menu[meal].products;

for (const product of neededProducts) {

let [productName, productQuantity] = product.split(/\s+/);

let quantityNeeded = Number(productQuantity);

if (!this.productsInStock.hasOwnProperty(productName) || this.productsInStock[productName] < quantityNeeded) {

return `For the time being, we cannot complete your order (${meal}), we are very sorry...`;

}

}

neededProducts.forEach((product) => {

let [productName, productQuantity] = product.split(/\s+/);

this.productsInStock[productName] -= Number(productQuantity);

});

let price = this.menu[meal].price;

this.budget += price;

return `Your order (${meal}) will be completed in the next 30 minutes and will cost you ${price}.`

}

}

# JS Advanced - Retake Exam: 18.11.2018

## 9.\*\*Warehouse – Unit Testing

class Warehouse {

get capacity() {

return this.\_capacity;

}

set capacity(givenSpace) {

if (typeof givenSpace === 'number' && givenSpace > 0) {

return this.\_capacity = givenSpace;

} else {

throw `Invalid given warehouse space`;

}

}

constructor(capacity) {

this.capacity = capacity;

this.availableProducts = {'Food': {}, 'Drink': {}};

}

addProduct(type, product, quantity) {

let addedQuantity = ((this.capacity - this.occupiedCapacity()) - quantity);

let output;

if (addedQuantity >= 0) {

if (this.availableProducts[type].hasOwnProperty(product) === false) {

this.availableProducts[type][product] = 0;

}

this.availableProducts[type][product] += quantity;

output = this.availableProducts[type];

} else {

throw `There is not enough space or the warehouse is already full`;

}

return output;

}

orderProducts(type) {

let output;

let sortedKeys = Object.keys(this.availableProducts[type])

.sort((a, b) => this.availableProducts[type][b] - this.availableProducts[type][a]);

let newObj = {};

for (let product of sortedKeys) {

if (newObj.hasOwnProperty(product) === false) {

newObj[product] = 0;

}

newObj[product] += this.availableProducts[type][product];

}

this.availableProducts[type] = newObj;

output = this.availableProducts[type];

return output;

}

occupiedCapacity() {

let output = 0;

let productsCount = Object.keys(this.availableProducts['Food']).length +

Object.keys(this.availableProducts['Drink']).length;

if (productsCount > 0) {

let quantityInStock = 0;

for (let type of Object.keys(this.availableProducts)) {

for (let product of Object.keys(this.availableProducts[type])) {

quantityInStock += this.availableProducts[type][product];

}

}

output = quantityInStock;

}

return output;

}

revision() {

let output = "";

if (this.occupiedCapacity() > 0) {

for (let type of Object.keys(this.availableProducts)) {

output += `Product type - [${type}]\n`;

for (let product of Object.keys(this.availableProducts[type])) {

output += `- ${product} ${this.availableProducts[type][product]}\n`;

}

}

} else {

output = 'The warehouse is empty';

}

return output.trim();

}

scrapeAProduct(product, quantity) {

let type = Object.keys(this.availableProducts).find(t => Object.keys(this.availableProducts[t]).includes(product));

let output;

if (type !== undefined) {

if (quantity <= this.availableProducts[type][product]) {

this.availableProducts[type][product] -= quantity;

} else {

this.availableProducts[type][product] = 0;

}

output = this.availableProducts[type];

} else {

throw `${product} do not exists`;

}

return output;

}

}

module.exports = {Warehouse};

// 100/100 in Judge but the second zero test didn't past - local all is OK

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

let Warehouse = require('../09Warehouse').Warehouse;

//In Judge must be paste without this above

describe('Warehouse', function () {

describe('constructor', function () {

it('should have correct capacity', function () {

let warehouse = new Warehouse(10);

expect(warehouse.capacity).to.be.equal(10);

});

it('should throw on negative capacity', function () {

expect(() => { new Warehouse(-10); }).to.throw('Invalid given warehouse space');

});

it('should throw on 0 capacity', function () {

expect(() => { new Warehouse(0); }).to.throw('Invalid given warehouse space');

});

it('should throw on string instead number for capacity', function () {

expect(() => { new Warehouse('10'); }).to.throw('Invalid given warehouse space');

});

});

describe('addProduct', function () {

it('should add product correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 9);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(1);

});

it('should add products correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 3);

warehouse.addProduct('Food', 'orange', 3);

warehouse.addProduct('Food', 'apple', 4);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(3);

});

it('should throw on not enough capacity for one product', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 11);

}).to.throw('There is not enough space or the warehouse is already full');

});

it('should throw on not enough capacity for more products', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 4);

warehouse.addProduct('Food', 'apple', 4);

warehouse.addProduct('Food', 'orange', 3);

}).to.throw('There is not enough space or the warehouse is already full');

});

});

describe('orderProducts', function () {

it('should order products for food with correct values', function () {

let warehouse = new Warehouse(1000);

warehouse.addProduct('Drink', 'tea', 10);

warehouse.addProduct('Food', 'bannana', 10);

warehouse.addProduct('Food', 'apple', 15);

warehouse.addProduct('Food', 'bannana', 30);

warehouse.addProduct('Drink', 'coffee', 15);

warehouse.addProduct('Food', 'orange', 10);

warehouse.addProduct('Drink', 'milk', 30);

let food = warehouse.orderProducts('Food');

let foodProducts = Object.keys(food);

let expectedFood = {

'bannana': 40,

'apple': 15,

'orange': 10

};

let expectedFoodProducts = Object.keys(expectedFood);

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

expect(foodProducts[i]).to.be.equal(expectedFoodProducts[i]);

}

});

it('should order products for drink with correct values', function () {

let warehouse = new Warehouse(1000);

warehouse.addProduct('Drink', 'tea', 10);

warehouse.addProduct('Food', 'bannana', 10);

warehouse.addProduct('Food', 'apple', 15);

warehouse.addProduct('Food', 'bannana', 30);

warehouse.addProduct('Drink', 'coffee', 15);

warehouse.addProduct('Food', 'orange', 10);

warehouse.addProduct('Drink', 'milk', 30);

let drink = warehouse.orderProducts('Drink');

let drinkProducts = Object.keys(drink);

let expectedDrink = {

'milk': 30,

'coffee': 15,

'tea': 10

};

let expectedDrinkProducts = Object.keys(expectedDrink);

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

expect(drinkProducts[i]).to.be.equal(expectedDrinkProducts[i]);

}

});

});

describe('occupiedCapacity', function () {

it('should return 0 capacity for empty space', function () {

let warehouse = new Warehouse(10);

expect(warehouse.occupiedCapacity()).to.be.equal(0);

});

it('should return correct capacity for full space with food', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with food and drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

});

describe('scrapeAProduct', function () {

it('should throw for non existing type', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.scrapeAProduct('orange', 1)

}).to.throw('orange do not exists');

});

it('should give correct result for less than available quantity', function(){

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

expect(JSON.stringify(warehouse.scrapeAProduct('bannana', 5))).to.be.equal('{"bannana":1,"apple":3}');

});

it('should give 0 for more than available quantity', function(){

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

expect(JSON.stringify(warehouse.scrapeAProduct('bannana', 7))).to.be.equal('{"bannana":0,"apple":3}');

});

});

describe('revision', function () {

it('should return correct message for empty warehouse', function () {

let warehouse = new Warehouse(10);

expect(warehouse.revision()).to.be.equal('The warehouse is empty');

});

it('should return correct value for warehouse with food', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- apple 3\n- orange 1\nProduct type - [Drink]';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

let expectedString = 'Product type - [Food]\nProduct type - [Drink]\n- milk 6\n- coffee 3\n- tea 1';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with food and drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- orange 1\nProduct type - [Drink]\n- tea 3';

expect(warehouse.revision()).to.be.equal(expectedString);

});

});

describe('constructor', function () {

it('should have correct capacity', function () {

let warehouse = new Warehouse(10);

expect(warehouse.capacity).to.be.equal(10);

});

it('should throw on negative capacity', function () {

expect(() => { new Warehouse(-10); }).to.throw('Invalid given warehouse space');

});

it('should throw on 0 capacity', function () {

expect(() => { new Warehouse(0); }).to.throw('Invalid given warehouse space');

});

it('should throw on string instead number for capacity', function () {

expect(() => { new Warehouse('10'); }).to.throw('Invalid given warehouse space');

});

});

describe('addProduct', function () {

it('should add product correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 9);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(1);

});

it('should add products correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 3);

warehouse.addProduct('Food', 'orange', 3);

warehouse.addProduct('Food', 'apple', 4);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(3);

});

it('should throw on not enough capacity for one product', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 11);

}).to.throw('There is not enough space or the warehouse is already full');

});

it('should throw on not enough capacity for more products', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 4);

warehouse.addProduct('Food', 'apple', 4);

warehouse.addProduct('Food', 'orange', 3);

}).to.throw('There is not enough space or the warehouse is already full');

});

});

describe('orderProducts', function () {

it('should order products for food with correct values', function () {

let warehouse = new Warehouse(1000);

warehouse.addProduct('Drink', 'tea', 10);

warehouse.addProduct('Food', 'bannana', 10);

warehouse.addProduct('Food', 'apple', 15);

warehouse.addProduct('Food', 'bannana', 30);

warehouse.addProduct('Drink', 'coffee', 15);

warehouse.addProduct('Food', 'orange', 10);

warehouse.addProduct('Drink', 'milk', 30);

let food = warehouse.orderProducts('Food');

let foodProducts = Object.keys(food);

let expectedFood = {

'bannana': 40,

'apple': 15,

'orange': 10

};

let expectedFoodProducts = Object.keys(expectedFood);

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

expect(foodProducts[i]).to.be.equal(expectedFoodProducts[i]);

}

});

});

describe('occupiedCapacity', function () {

let warehouse;

beforeEach(function(){

warehouse = new Warehouse(10);

});

it('should return 0 capacity for empty space', function () {

expect(warehouse.occupiedCapacity()).to.be.equal(0);

});

it('should return correct capacity for full space with food', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with drink', function () {

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with food and drink', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

});

describe('scrapeAProduct', function () {

let warehouse;

beforeEach(function() {

warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

});

it('should throw for non existing type', function () {

expect(() => {

warehouse.scrapeAProduct('orange', 1)

}).to.throw('orange do not exists');

});

it('should give correct result for less than available quantity', function(){

expect(JSON.stringify(warehouse.scrapeAProduct('banana', 3))).to.be.equal('{"banana":1,"apple":1}');

});

it('should give 0 for more than available quantity', function(){

expect(JSON.stringify(warehouse.scrapeAProduct('banana', 5))).to.be.equal('{"banana":0,"apple":1}');

});

});

describe('revision', function () {

let warehouse;

beforeEach(function(){

warehouse = new Warehouse(10);

});

it('should return correct message for empty warehouse', function () {

expect(warehouse.revision()).to.be.equal('The warehouse is empty');

});

it('should return correct value for warehouse with food', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- apple 3\n- orange 1\nProduct type - [Drink]';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with drink', function () {

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

let expectedString = 'Product type - [Food]\nProduct type - [Drink]\n- milk 6\n- coffee 3\n- tea 1';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with food and drink', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- orange 1\nProduct type - [Drink]\n- tea 3';

expect(warehouse.revision()).to.be.equal(expectedString);

});

});

});

//81/100 in Judge

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

let Warehouse = require('../09Warehouse').Warehouse;

//In Judge must be paste without this above

describe('Warehouse', function () {

describe('constructor', function () {

it('should have property capacity', function () {

let warehouse = new Warehouse(10);

expect(warehouse.hasOwnProperty('\_capacity')).to.be.true;

})

it('should have correct capacity', function () {

let warehouse = new Warehouse(10);

expect(warehouse.capacity).to.be.equal(10);

});

it('should have property availableProducts', function () {

let warehouse = new Warehouse(10);

expect(warehouse.hasOwnProperty('availableProducts')).to.be.true;

});

it('should have property availableProducts with propery Food', function () {

let warehouse = new Warehouse(10);

expect(warehouse.availableProducts.hasOwnProperty('Food')).to.be.true;

});

it('should have property availableProducts with propery Drink', function () {

let warehouse = new Warehouse(10);

expect(warehouse.availableProducts.hasOwnProperty('Drink')).to.be.true;

});

it('should throw on negative capacity', function () {

expect(() => { new Warehouse(-10); }).to.throw('Invalid given warehouse space');

});

it('should throw on 0 capacity', function () {

expect(() => { new Warehouse(0); }).to.throw('Invalid given warehouse space');

});

it('should throw on string instead number for capacity', function () {

expect(() => { new Warehouse('10'); }).to.throw('Invalid given warehouse space');

});

});

describe('addProduct', function () {

it('should add product correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 9);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(1);

});

it('should add products correctly', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 3);

warehouse.addProduct('Food', 'orange', 3);

warehouse.addProduct('Food', 'apple', 4);

expect(Object.keys(warehouse.availableProducts['Food']).length).to.be.equal(3);

});

it('should throw on not enough capacity for one product', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 11);

}).to.throw('There is not enough space or the warehouse is already full');

});

it('should throw on not enough capacity for more products', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 4);

warehouse.addProduct('Food', 'apple', 4);

warehouse.addProduct('Food', 'orange', 3);

}).to.throw('There is not enough space or the warehouse is already full');

});

// it('shold throw TypeError for not correct type', function(){

// expect(() => {

// let warehouse = new Warehouse(10);

// warehouse.addProduct('food', 'bannana', 11);

// }).to.throw(TypeError, "Cannot read property 'hasOwnProperty' of undefined");

// });

});

describe('orderProducts', function () {

it('should order products for food with correct values', function () {

let warehouse = new Warehouse(1000);

warehouse.addProduct('Drink', 'tea', 10);

warehouse.addProduct('Food', 'bannana', 10);

warehouse.addProduct('Food', 'apple', 15);

warehouse.addProduct('Food', 'bannana', 30);

warehouse.addProduct('Drink', 'coffee', 15);

warehouse.addProduct('Food', 'orange', 10);

warehouse.addProduct('Drink', 'milk', 30);

let food = warehouse.orderProducts('Food');

let foodProducts = Object.keys(food);

let expectedFood = {

'bannana': 40,

'apple': 15,

'orange': 10

};

let expectedFoodProducts = Object.keys(expectedFood);

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

expect(foodProducts[i]).to.be.equal(expectedFoodProducts[i]);

}

});

it('should order products for drink with correct values', function () {

let warehouse = new Warehouse(1000);

warehouse.addProduct('Drink', 'tea', 10);

warehouse.addProduct('Food', 'bannana', 10);

warehouse.addProduct('Food', 'apple', 15);

warehouse.addProduct('Food', 'bannana', 30);

warehouse.addProduct('Drink', 'coffee', 15);

warehouse.addProduct('Food', 'orange', 10);

warehouse.addProduct('Drink', 'milk', 30);

let drink = warehouse.orderProducts('Drink');

let drinkProducts = Object.keys(drink);

let expectedDrink = {

'milk': 30,

'coffee': 15,

'tea': 10

};

let expectedDrinkProducts = Object.keys(expectedDrink);

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

expect(drinkProducts[i]).to.be.equal(expectedDrinkProducts[i]);

}

});

});

describe('occupiedCapacity', function () {

it('should return 0 capacity for empty space', function () {

let warehouse = new Warehouse(10);

expect(warehouse.occupiedCapacity()).to.be.equal(0);

});

it('should return correct capacity for full space with food', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with food and drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

});

describe('occupiedCapacity', function () {

let warehouse;

beforeEach(function(){

warehouse = new Warehouse(10);

});

it('should return 0 capacity for empty space', function () {

expect(warehouse.occupiedCapacity()).to.be.equal(0);

});

it('should return correct capacity for full space with food', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with drink', function () {

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

it('should return correct capacity for full space with food and drink', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

expect(warehouse.occupiedCapacity()).to.be.equal(10);

});

});

describe('revision', function () {

it('should return correct message for empty warehouse', function () {

let warehouse = new Warehouse(10);

expect(warehouse.revision()).to.be.equal('The warehouse is empty');

});

it('should return correct value for warehouse with food', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- apple 3\n- orange 1\nProduct type - [Drink]';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

let expectedString = 'Product type - [Food]\nProduct type - [Drink]\n- milk 6\n- coffee 3\n- tea 1';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with food and drink', function () {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- orange 1\nProduct type - [Drink]\n- tea 3';

expect(warehouse.revision()).to.be.equal(expectedString);

});

});

describe('revision', function () {

let warehouse;

beforeEach(function(){

warehouse = new Warehouse(10);

});

it('should return correct message for empty warehouse', function () {

expect(warehouse.revision()).to.be.equal('The warehouse is empty');

});

it('should return correct value for warehouse with food', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- apple 3\n- orange 1\nProduct type - [Drink]';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with drink', function () {

warehouse.addProduct('Drink', 'milk', 6);

warehouse.addProduct('Drink', 'coffee', 3);

warehouse.addProduct('Drink', 'tea', 1);

let expectedString = 'Product type - [Food]\nProduct type - [Drink]\n- milk 6\n- coffee 3\n- tea 1';

expect(warehouse.revision()).to.be.equal(expectedString);

});

it('should return correct value for warehouse with food and drink', function () {

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Drink', 'tea', 3);

warehouse.addProduct('Food', 'orange', 1);

let expectedString = 'Product type - [Food]\n- bannana 6\n- orange 1\nProduct type - [Drink]\n- tea 3';

expect(warehouse.revision()).to.be.equal(expectedString);

});

});

describe('scrapeAProduct', function () {

// it('should return string for non existing type', function () {

// let warehouse = new Warehouse(10);

// warehouse.addProduct('Food', 'bannana', 6);

// warehouse.addProduct('Food', 'apple', 3);

// expect(warehouse.scrapeAProduct('orange', 1)).to.be.equal('orange do not exist');

// });

it('should throw for non existing type', function () {

expect(() => {

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

warehouse.scrapeAProduct('orange', 1)

}).to.throw('orange do not exist');

});

it('should give correct result for less than available quantity', function(){

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

expect(JSON.stringify(warehouse.scrapeAProduct('bannana', 5))).to.be.equal('{"bannana":1,"apple":3}');

});

it('should give 0 for more than available quantity', function(){

let warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

expect(JSON.stringify(warehouse.scrapeAProduct('bannana', 7))).to.be.equal('{"bannana":0,"apple":3}');

});

});

describe('scrapeAProduct', function () {

let warehouse;

beforeEach(function(){

warehouse = new Warehouse(10);

warehouse.addProduct('Food', 'bannana', 6);

warehouse.addProduct('Food', 'apple', 3);

});

// it('should return string for non existing type', function () {

// expect(warehouse.scrapeAProduct('orange', 1)).to.be.equal('orange do not exist');

// });

it('should throw for non existing type', function () {

expect(() => {

warehouse.scrapeAProduct('orange', 1)

}).to.throw('orange do not exist');

});

it('should give correct result for less than available quantity', function(){

expect(JSON.stringify(warehouse.scrapeAProduct('bannana', 5))).to.be.equal('{"bannana":1,"apple":3}');

});

it('should give 0 for more than available quantity', function(){

expect(JSON.stringify(warehouse.scrapeAProduct('bannana', 7))).to.be.equal('{"bannana":0,"apple":3}');

});

});

});

// 100/100 in Judge but the second zero test didn't past

describe('test initialization', function () {

it('should have correct capacity', function () {

let warehouse = new Warehouse(5);

expect(warehouse.capacity).to.be.equal(5);

});

it('should throw error on negative capacity', function () {

expect(function () {

let warehouse = new Warehouse(-1);

}).to.throw('Invalid given warehouse space')

});

it('should throw error on zero capacity', function () {

expect(function () {

let warehouse = new Warehouse(0);

}).to.throw('Invalid given warehouse space')

});

it('should throw error on zero capacity', function () {

expect(function () {

let warehouse = new Warehouse("1");

}).to.throw('Invalid given warehouse space')

});

it('should throw error on not a number', function () {

expect(function () {

let warehouse = new Warehouse("ddsfs");

}).to.throw('Invalid given warehouse space')

});

});

describe('addProduct(type, product, quantity) method test', function () {

it('should add product', function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

expect(Object.keys(warehouse.availableProducts["Food"]).length).to.be.equal(2);

});

it('should throw error on not enough capacity', function () {

expect(function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 10);

}).to.throw('There is not enough space or the warehouse is already full');

});

it('should throw error on not enough capacity', function () {

expect(function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 5);

warehouse.addProduct("Food", "banana", 1);

}).to.throw('There is not enough space or the warehouse is already full');

});

it('should throw error on not correct type', function () {

expect(function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("food", "banana", 3);

}).to.throw(TypeError, "Cannot read property 'hasOwnProperty' of undefined");

});

});

describe('orderProducts(type) method test', function () {

it('orderProducts with correct values', function () {

let test = new Warehouse(3000);

test.addProduct('Drink', 'coke', 30);

test.addProduct('Food', 'sliva', 12);

test.addProduct('Food', 'burger', 16);

test.addProduct('Food', 'banana', 55);

test.addProduct('Drink', 'cappy', 77);

let output = test.orderProducts('Food');

let expectedOutput = {

'banana': 55,

'burger': 16,

'sliva': 12

}

let expectedProducts = Object.keys(expectedOutput);

let outputProducts = Object.keys(output);

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

expect(outputProducts[i]).to.be.equal(expectedProducts[i]);

}

});

});

describe('occupiedCapacity() method test', function () {

it('should return correct capacity', function () {

let warehouse = new Warehouse(5);

expect(warehouse.occupiedCapacity()).to.be.equal(0);

});

it('should return correct capacity', function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

expect(warehouse.occupiedCapacity()).to.be.equal(5);

});

it('should return correct capacity', function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Drink", "juice", 4);

warehouse.addProduct("Drink", "water", 1);

expect(warehouse.occupiedCapacity()).to.be.equal(5);

});

it('should return correct capacity', function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Drink", "juice", 4);

warehouse.addProduct("Food", "apple", 1);

expect(warehouse.occupiedCapacity()).to.be.equal(5);

});

});

describe('scrapeAProduct(product, quantity) method test', function () {

it('throw error for non existing type', function () {

expect(function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

warehouse.scrapeAProduct('tomato', 1)

}).to.throw('tomato do not exists');

});

it('throw give correct result for less than available quantity', function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

expect(JSON.stringify(warehouse.scrapeAProduct('banana', 3))).to.be.equal('{"banana":1,"apple":1}');

});

it('throw give 0 for more than available quantity', function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

expect(JSON.stringify(warehouse.scrapeAProduct('banana', 5))).to.be.equal('{"banana":0,"apple":1}');

});

});

describe('revision() method test', function () {

it('should give correct message for empty warehouse', function () {

let warehouse = new Warehouse(5);

expect(warehouse.revision()).to.be.equal("The warehouse is empty")

});

it('should ', function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

expect(warehouse.revision()).to.be.equal("Product type - [Food]\n- banana 4\n- apple 1\nProduct type - [Drink]");

});

it('should ', function () {

let warehouse2 = new Warehouse(5);

warehouse2.addProduct("Food", "banana", 4);

warehouse2.addProduct("Drink", "water", 1);

expect(warehouse2.revision()).to.be.equal("Product type - [Food]\n- banana 4\nProduct type - [Drink]\n- water 1");

});

it('should ', function () {

let warehouse3 = new Warehouse(5);

warehouse3.addProduct("Drink", "juice", 4);

warehouse3.addProduct("Drink", "water", 1);

expect(warehouse3.revision()).to.be.equal("Product type - [Food]\nProduct type - [Drink]\n- juice 4\n- water 1");

});

});

describe('test initialization', function () {

it('should have correct capacity', function () {

let warehouse = new Warehouse(5);

expect(warehouse.capacity).to.be.equal(5);

});

it('should throw error on negative capacity', function () {

expect(function () {

let warehouse = new Warehouse(-1);

}).to.throw('Invalid given warehouse space')

});

it('should throw error on zero capacity', function () {

expect(function () {

let warehouse = new Warehouse(0);

}).to.throw('Invalid given warehouse space')

});

it('should throw error on zero capacity', function () {

expect(function () {

let warehouse = new Warehouse("1");

}).to.throw('Invalid given warehouse space')

});

it('should throw error on not a number', function () {

expect(function () {

let warehouse = new Warehouse("ddsfs");

}).to.throw('Invalid given warehouse space')

});

});

describe('addProduct(type, product, quantity) method test', function () {

it('should add product', function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

expect(Object.keys(warehouse.availableProducts["Food"]).length).to.be.equal(2);

});

it('should throw error on not enough capacity', function () {

expect(function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 10);

}).to.throw('There is not enough space or the warehouse is already full');

});

it('should throw error on not enough capacity', function () {

expect(function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 5);

warehouse.addProduct("Food", "banana", 1);

}).to.throw('There is not enough space or the warehouse is already full');

});

it('should throw error on not correct type', function () {

expect(function () {

let warehouse = new Warehouse(5);

warehouse.addProduct("food", "banana", 3);

}).to.throw(TypeError, "Cannot read property 'hasOwnProperty' of undefined");

});

});

describe('orderProducts(type) method test', function () {

it('orderProducts with correct values', function () {

let test = new Warehouse(3000);

test.addProduct('Drink', 'coke', 30);

test.addProduct('Food', 'sliva', 12);

test.addProduct('Food', 'burger', 16);

test.addProduct('Food', 'banana', 55);

test.addProduct('Drink', 'cappy', 77);

let output = test.orderProducts('Food');

let expectedOutput = {

'banana': 55,

'burger': 16,

'sliva': 12

}

let expectedProducts = Object.keys(expectedOutput);

let outputProducts = Object.keys(output);

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

expect(outputProducts[i]).to.be.equal(expectedProducts[i]);

}

});

});

describe('occupiedCapacity() method test', function () {

let warehouse;

beforeEach(function () {

warehouse = new Warehouse(5);

});

it('should return correct capacity', function () {

expect(warehouse.occupiedCapacity()).to.be.equal(0);

});

it('should return correct capacity', function () {

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

expect(warehouse.occupiedCapacity()).to.be.equal(5);

});

});

describe('scrapeAProduct(product, quantity) method test', function () {

let warehouse;

beforeEach(function () {

warehouse = new Warehouse(5);

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

});

it('throw error for non existing type', function () {

expect(function () {

warehouse.scrapeAProduct('tomato', 1)

}).to.throw('tomato do not exists');

});

it('throw give correct result for less than available quantity', function () {

expect(JSON.stringify(warehouse.scrapeAProduct('banana', 3))).to.be.equal('{"banana":1,"apple":1}');

});

it('throw give 0 for more than available quantity', function () {

expect(JSON.stringify(warehouse.scrapeAProduct('banana', 5))).to.be.equal('{"banana":0,"apple":1}');

});

});

describe('revision() method test', function () {

let warehouse;

beforeEach(function () {

warehouse = new Warehouse(5);

});

it('should give correct message for empty warehouse', function () {

expect(warehouse.revision()).to.be.equal("The warehouse is empty")

});

it('should ', function () {

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Food", "apple", 1);

expect(warehouse.revision()).to.be.equal("Product type - [Food]\n- banana 4\n- apple 1\nProduct type - [Drink]");

});

it('should ', function () {

warehouse.addProduct("Food", "banana", 4);

warehouse.addProduct("Drink", "water", 1);

expect(warehouse.revision()).to.be.equal("Product type - [Food]\n- banana 4\nProduct type - [Drink]\n- water 1");

});

it('should ', function () {

warehouse.addProduct("Drink", "juice", 4);

warehouse.addProduct("Drink", "water", 1);

expect(warehouse.revision()).to.be.equal("Product type - [Food]\nProduct type - [Drink]\n- juice 4\n- water 1");

});

});