# JS Advanced: Exam 18.10.2018

## Problem 2. Warehouse (Unit Testing)

You are given the following **JavaScript class**:

|  |
| --- |
| **Warehouse.js** |
| **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;  } } |

### Functionality

An **instance** of the **Vacation** class should support the following operations:

If the **constructor** gets a **negative number** or **0 should throw a string:  
"Invalid given warehouse space"**

#### addProduct(type, product, quantity)

**Adds** the given product if there is space in the warehouse and **return the** . In these cases when the product is added more than 1 time, the quantity should be **sum**. When there is **no place** for the current product, you should **throw** a string that says: **"There is not enough space or the warehouse is already full"**

#### orderProducts(type)

**Sorts all products** of a given **type in descending order** by the **quantity.**

#### occupiedCapacity()

**Returns** a number, which represents the **already occupied** place in the warehouse.

#### revision()

Returns a string in which we print all products of each type, into the following format:

**'Product type - [Food]'**

**- {product} {quantity}**

**- {product} quantity  
...  
...**

**'Product type - [Drink]'**

**- {product} {quantity}**

**- {product} quantity  
...  
...  
If there is not at least 1 product in the warehouse** we **return** the string:   
**'The Warehouse is empty'**

#### scrapeAProduct(product, quantity)

If the given **product exists** we reduce his quantity, otherwise we **reset it.** If we cannot find the given product we **return** **the** **string**: **'{product} do not exists'**

### TODO

Using **Mocha** and **Chai** write **JS unit tests** to test the entire functionality of the **Vacation** class. You may use the following code as a template:

**Submit** only your **describe()** statements..

|  |
| --- |
| describe(**"*TODO* …"**, **function**() {  ***it***(**"*TODO …*"**, **function**() {  *//* ***TODO:*** …  });  *//* ***TODO:*** …  });  describe(**"*TODO* …"**, **function**() {  ***it***(**"*TODO …*"**, **function**() {  *//* ***TODO:*** …  });  *//* ***TODO:*** …  });  ...  ... |

Don't forget to require the chai library!