# JS Advanced: Exam Preparation 1

# Problem 1. Furniture Store

**Link in Judge:** [**https://judge.softuni.org/Contests/Practice/Index/3089#0**](https://judge.softuni.org/Contests/Practice/Index/3089#0)

**Environment Specifics**

Please, be aware that every JS environment may **behave differently** when executing code. Certain things that work in the browser are not supported in **Node.js**, which is the environment used by **Judge**.

The following actions are **NOT** supported:

* **.forEach()** with **NodeList** (returned by **querySelector()** and **querySelectorAll()**)
* **.forEach()** with **HTMLCollection** (returned by **getElementsByClassName()** and **element.children**)
* Using the **spread-operator** (**...**) to convert a **NodeList** into an array
* **append()** in Judge (use only **appendChild()**)
* **replaceWith()** in Judge
* **replaceAll()** in Judge
* **closest()** in Judge

If you want to perform these operations, you may use **Array.from()** to first convert the collection into an array.

**Use the provided skeleton to solve this problem.**

**Note**: You **can't** and you have no permission to **change** directly the given HTML code (index.html file).

**Write the missing JavaScript code** to make the **Furniture** **Store** work as expected:



**Your Task**

* A**ll fields (model, year, description, and price)** are **filled with the correct input**
  + **Model and description** are **non**-**empty** **strings**
  + **Year** and **Price** need to be **positive** **numbers**
  + **All fields must be filled**

1. **Getting the furniture information**



* When you click the “Add” button, the information from the input fields must be added to the table and then clear input fields.
* The table contains **Model, Price of furniture** and **Actions** - **[More information], [Buy it]**. **The price** must be **rounded** to the **second** digit after the decimal point.



**Each furniture** must be appended to **"furniture-list"** and look like the picture below: 

Each piece of furniture has the main information line **(Model, Price)** and an additional information line. The additional information line stores **the description and year** of manufacture of the furniture **(hidden until the "More info" button is pressed**)**.**

When the **"More Info"** button is clicked, change button text from **"More Info"** to **"Less Info"** and style display of **"class = hide"** from **"none "** to **"contents".** The second **<td>** must-have attribute **colspan** with value **3.** When click **"Less Info"** button is clicked, change button text from **"Less Info"** to **"More Info"** and style from **"contents "** to **"none".**





When the **"Buy it"** button is clicked, should have the following **functionality:**

* **The current furniture** must be **removed from the row** on the table
* You need to **change** the **total** profit in **the store.** Take the element with class **"total-price"** **and increase the current total price** **with** the **price of the furniture**.



# Problem 2. Camping

**Link in Judge:** [**https://judge.softuni.org/Contests/Practice/Index/3235#1**](https://judge.softuni.org/Contests/Practice/Index/3235#1)



Write a **class Summer camp**, which supports the described functionality below.

**Functionality**

**Constructor**

Should have these **4** properties:

* **organizer - string**
* **location - string**
* **priceForTheCamp - {"child": 150, "student": 300, "collegian": 500}**
* **listOfParticipants - empty array**

**At the initialization** of the **SummerCamp** class, the **constructor** accepts the **organizer** and **location.** The **priceForTheCamp** is an **object**, the **submitted values** are by **default** and represent the price for the stay in the camp depending on the **condition** of the participant **("child", "student", "collegian")**.

### registerParticipant (name, condition, money)

This method register participant to the camping. The methodaccepts 3 arguments:

* **name (string);**
* **condition (string);**
* **money (number);**
* If the given **condition** of participants, is not present in **priceForTheCamp** object with the specified default values **("child", "student", "collegian")**, an error with the following message should be **thrown**:

**"Unsuccessful registration at the camp."**

* If the **name** of the current participant is already present in **listOfParticipants** array, **return** the following message:

**`The {name} is already registered at the camp.`**

* If the submitted **money** is less than the **price** for the stay in the camp (the **price** is determined by the **priceForTheCamp** object, depending on the **condition** of the participant), **return** the following message:

**`The money is not enough to pay the stay at the camp.`**

* Otherwise, should **add** the participant, with properties: **{name, condition, power: default 100,** **wins: default 0}** to the **listOfParticipants array** and **return**:

**`The {name} was successfully registered.`**

### unregisterParticipant (name)

This method removes a participant from the camping. The methodaccepts 1 argument:

* **name (string)**;
* If the **name** of the current participant is not present in the **listOfParticipants** array, an error with the following message should be **thrown**:

**`The {name} is not registered in the camp.`**

* **Otherwise,** this function should **remove** the participant from the **listOfParticipants** arrayand **return:**

**`The {name} removed successfully.`**

### timeToPlay (typeOfGame, participant1, participant2)

Method can take 2 or 3 arguments depending on the type of game:

* **typeOfGame (string);**
* **participant1 - name(string);**
* **participant2 - name(string) - optional;**
* There are two possible types of games:
* **WaterBalloonFights** -> you will get **two** players**.**

Example **-> timeToPlay ("WaterBalloonFights", "Petar", "John")**

**Note:** The **condition** of the participants must match **(**Example: **"Petar" - "child" and "John" - "child")**

* **Battleship** -> you will get **one** player**.**

Example **-> timeToPlay ("Battleship", "Petar")**

* If any of the submitted participants **names** are not presentin the **listOfParticipants** array, an error with the following message should be **thrown**:

**`Invalid entered name/s.`**

* If two names are submitted, check that the participants' **condition** matches, if not matched, an error with the following message should be **thrown**:

**`Choose players with equal condition.`**

* If the type of game is **Battleship** increase the **power** property of the **participant** by a **value** of **20**, and **return** the message:

**`The {name}** **successfully completed the game {typeOfGame}.`**

* If the type of game is **WaterBalloonFights,** you must check whether the value of the **power** of one participant **is greater** than the value of the **power** of the **other** participant, and in this case increase the value of the **wins** property **by one** per **winner** (with the **bigger power**), and **return** the following message:

**`The {name} is winner in the game {typeOfGame}.`**

**Note:** The **{name}** is the name of the winner in this game.

* Otherwise, the function **returns** the message:

**`There is no winner.`**

**toString ()**

* At the first line return:

**`{organizer} will take {numberOfParticipants} participants on camping to {location}`**

* On the lines, display information about each **participant, sorted** in **descending** order by their **wins** in the following format:

**`{name} - {condition} - {power} - {wins}`**

### Examples

|  |
| --- |
| **Input 1** |
| **const** **summerCamp** = **new** **SummerCamp("Jane Austen", "Pancharevo Sofia 1137, Bulgaria");**  console.log(summerCamp.registerParticipant("Petar Petarson", "student", 200));  console.log(summerCamp.registerParticipant("Petar Petarson", "student", 300));  console.log(summerCamp.registerParticipant("Petar Petarson", "student", 300));  console.log(summerCamp.registerParticipant("Leila Wolfe", "childd", 200)); |

|  |
| --- |
| **Output 1** |
| The money is not enough to pay the stay at the camp.  The Petar Petarson was successfully registered.  The Petar Petarson is already registered at the camp.  **Uncaught Error:** Unsuccessful registration at the camp. |

|  |
| --- |
| **Input 2** |
| **const** **summerCamp** = **new** **SummerCamp("Jane Austen", "Pancharevo Sofia 1137, Bulgaria");**  console.log(summerCamp.registerParticipant("Petar Petarson", "student", 300));  console.log(summerCamp.unregisterParticipant("Petar"));  console.log(summerCamp.unregisterParticipant("Petar Petarson")); |

|  |
| --- |
| **Output 2** |
| The Petar Petarson was successfully registered.  **Uncaught Error:** The Petar is not registered in the camp.  The Petar Petarson removed successfully. |

|  |
| --- |
| **Input 3** |
| **const** **summerCamp** = **new** **SummerCamp("Jane Austen", "Pancharevo Sofia 1137, Bulgaria");**  console.log(summerCamp.registerParticipant("Petar Petarson", "student", 300));  console.log(summerCamp.timeToPlay("Battleship", "Petar Petarson"));  console.log(summerCamp.registerParticipant("Sara Dickinson", "child", 200));  console.log(summerCamp.timeToPlay("WaterBalloonFights", "Petar Petarson", "Sara Dickinson"));  console.log(summerCamp.registerParticipant("Dimitur Kostov", "student", 300));  console.log(summerCamp.timeToPlay("WaterBalloonFights", "Petar Petarson", "Dimitur Kostov")); |

|  |
| --- |
| **Output 3** |
| The Petar Petarson was successfully registered.  The Petar Petarson successfully completed the game Battleship.  The Sara Dickinson was successfully registered.  **Uncaught Error:** Choose players with equal condition.  The Dimitur Kostov was successfully registered.  The Petar Petarson is winner in the game WaterBalloonFights. |

|  |
| --- |
| **Input 4** |
| **const** **summerCamp** = **new** **SummerCamp("Jane Austen", "Pancharevo Sofia 1137, Bulgaria");**  console.log(summerCamp.registerParticipant("Petar Petarson", "student", 300));  console.log(summerCamp.timeToPlay("Battleship", "Petar Petarson"));  console.log(summerCamp.registerParticipant("Sara Dickinson", "child", 200));  console.log(summerCamp.timeToPlay("WaterBalloonFights", "Petar Petarson", "Sara Dickinson"));  console.log(summerCamp.registerParticipant("Dimitur Kostov", "student", 300));  console.log(summerCamp.timeToPlay("WaterBalloonFights", "Petar Petarson", "Dimitur Kostov"));  console.log(summerCamp.toString()); |

|  |
| --- |
| **Output 4** |
| The Petar Petarson was successfully registered.  The Petar Petarson successfully completed the game Battleship.  The Sara Dickinson was successfully registered.  **Uncaught Error:** Choose players with equal condition.  The Dimitur Kostov was successfully registered.  The Petar Petarson is winner in the game WaterBalloonFights.  Jane Austen will take 3 participants on camping to Pancharevo Sofia 1137, Bulgaria  Petar Petarson - student - 120 - 1  Sara Dickinson - child - 100 - 0  Dimitur Kostov - student - 100 - 0 |

# Problem 3. Repository

**Link in Judge:** [**https://judge.softuni.org/Contests/Practice/Index/2819#1**](https://judge.softuni.org/Contests/Practice/Index/2819#1)

Use the provided Repository **class** to solve this problem.

### Your Task

Using **Mocha** and **Chai** write **JS Unit Tests** to test the entire functionality of the Repository **class**. Make sure instances of it have all the required functionality and validation. You may use the following code as a template:

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

### Functionality

solution.js defines a **class** that contains information about a **Repository class**. An **instance** of the class should support the following operations:

* **Instantiation** with **one** **parameter** - The **props** parameter, which is used to **validate** entities added to the repository and is an object**,** and **additional properties** called data ( **Map** that holds added entities).
* Getter count – returns the number of stored entities
* Function add(entity) – adds an entity to the data; if successful Store entities in a Map where the key is the ID and the value is the entity and returns the resulting ID. Before an entity is **added** to the repository, it should be **validated** against the props object – it needs to have all of the properties that the props object has and their values must be of the specified type. If **any** property is **missing**, you should **throw** an **Error** with the message: "**Property {propName} is missing from the entity!**". If the property is present but is of **incorrect** type, **throw** a **TypeError** with the message "**Property {propertyName} is of incorrect type!**"**.**
* Function getId(id) – returns the entity with given ID
* Function update(id, newEntity) – replaces the entity with the given id with the new entity. If the id does **not** exist in the **data** throw an **Error** with the message "**Entity with id: {id} does not exist!**". Validate the **new** entity with the **same** validations and **replace** the old one with the new one.
* Function del(id) – deletes an entity by the given id. If the id does **not** exist in the **data** throw an **Error** with the message "**Entity with id: {id} does not exist!**".

### Examples

|  |
| --- |
| Sample Code Usage |
| let properties = {      name: "string",      age: "number",      birthday: "object"  };  let repository = new Repository(properties);  let entity = {      name: "Pesho",      age: 22,      birthday: new Date(1998, 0, 7)  };  repository.add(entity);  repository.add(entity);  console.log(repository.getId(0));  console.log(repository.getId(1));  entity = {      name: 'Gosho',      age: 22,      birthday: new Date(1998, 0, 7)  };  repository.update(1, entity);  console.log(repository.getId(1));  repository.del(0);  console.log(repository.count); |
| Corresponding Output |
| **{ name: 'Pesho', age: 22, birthday: 1998-01-06T22:00:00.000Z }**  **{ name: 'Pesho', age: 22, birthday: 1998-01-06T22:00:00.000Z }**  **{ name: 'Gosho', age: 22, birthday: 1998-01-06T22:00:00.000Z }**  **1** |