More Exercise: Classes

1. Instance Validation

Write a class for a checking account that validates it's created with valid parameters. A **`CheckingAccount`** has a **`clientId`**, **`email`**, **`firstName`**, **`lastName`**. Each parameter must meet specific requirements:

- **`clientId`** - Must be a string representing a **6-digit number**; if invalid, throw a **`TypeError`** with the message "**`Client ID must be a 6-digit number`**"
- **`email`** - Must contain at least one **alphanumeric character**, followed by the **@** symbol, followed by **one** or **more** letters or periods; all letters must be **Latin**; if invalid, throw a **`TypeError`** with message "**`Invalid e-mail`**"
- **`firstName`**, **`lastName`** - Must be at least **3** and at most **20** characters long, containing **only** Latin letters;
    - If the **length** is invalid, throw a **`TypeError`** with message:

      `"{First/Last} name must be between 3 and 20 characters long"`
    - If invalid **characters** are used, throw a **`TypeError`** with message:
      `"{First/Last} name must contain only Latin characters"` (replace **`First/Last`** with the relevant word)

All checks must happen in the **order** in which **they are listed** - if more than one parameter is **invalid**, throw an error for the first encountered. Note that **error messages** must be **exact**.

Examples

Sample Input
`let acc = new CheckingAccount('1314', 'ivan@some.com', 'Ivan', Smith)`
Output
`TypeError: Client ID must be a 6-digit number`

Sample Input
`let acc = new CheckingAccount('131455', 'ivan@', 'Ivan', Smith)`
Output
`TypeError: Invalid e-mail`

Sample Input
`let acc = new CheckingAccount('131455', 'ivan@some.com', 'I', Smith)`
Output
`TypeError: First name must be between 3 and 20 characters long`

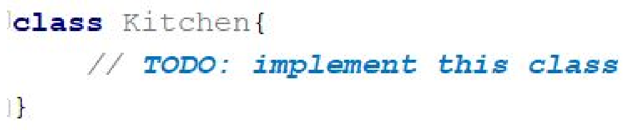
Sample Input
`let acc = new CheckingAccount('131455', 'ivan@some.com', 'Ivan', 'Sm1th')`
Output
`TypeError: "Last name must contain only Latin characters`

What to submit?

You are only required to submit the **CheckingAccount class**. No need to include the codes from the example above.

Class Signature: class **CheckingAccount**

2. Kitchen



```
class Kitchen{
    // TODO: implement this class
}
```

Write a class **`Kitchen`** which has the following functionality:

Constructor

Should have 4 properties:

- **`budget`**
- **`menu`**
- **`productsInStock`**
- **`actionsHistory`**

At initialization of the **`Kitchen`** class, the constructor accepts **only** the **budget!** The rest of the properties must be **empty**!

Methods:

- **`LoadProducts()`**
- Accept 1 property **`products`** (**array from strings**).
    - **Every element** into this array is information about product **in format**: **`"{productName} {productQuantity} {productPrice}"`**
    - They are separated by a **single space**
      Example: ["**`Banana 10 5`**", "**`Strawberries 50 30`**", "**`Honey 5 50`**"**...**]
- This method **appends products** into our products in stock (**`productsInStock`**) under the following circumstances:
    - **If the budget allows us to buy the current product**, we add it to **productsInStock** keeping **the name** and **quantity** of **the meal** and we **deduct the price of the product** from **our budget.** If the current product already exists into **productsInStock** just add the new quantity
    - And finally, **whether or not** we have **added** a product to stock or **not**, we **record** our **action** in the **actionsHistory**:
        - If we **were able to add** the current product:

`"Successfully loaded {productQuantity} {productName}"`

- If we **not**:

`"There was not enough money to load {productQuantity} {productName}"`

- This method must **return all actions joined by a new line!**

AddToMenu()

- Accept 3 properties **`meal`** (string)**, `needed products`** (array from strings) and **`price`** (number).
    - Every element into **needed products** is in format:
      `"{productName} {productQuantity}"`
    - They are separated by a **single space**!
- This method **appends a new meal** into our **menu and returns** the following message:
  `"Great idea! Now with the {meal} we have {the number of all means in the menu} meals in the menu, other ideas?"`
- **If** we **do not have** the **given meal** into our **menu**, we added it **keeping all** that we are given as information. Otherwise if we already have this meal print the **message**:
  `" The {meal} is already in our menu, try something different."`

ShowTheMenu()

- This method just **prints all meals** from our **menu separated by a new line** in format:

```
{meal} - $ {meal price}
{meal} - $ {meal price}
{meal} - $ {meal price}
...
```

At the end **trim the result!**

- If our menu **is empty**, just print the **message**:
  `"Our menu is not ready yet, please come later..."`

MakeTheOrder()

- Accept 1 property **meal** (string).
- This method **searches the menu** for a **certain meal**.
    - If **we do not have** the **given meal**, print the following **message**:
      `"There is not {meal} yet in our menu, do you want to order something else?"`
    - **Otherwise** if we **have this `meal`** in **the menu**, we need to check if we have the **needed products** to make it! If we **do not have all needed products** for this meal, print the following **message**:
      `"For the time being, we cannot complete your order ({meal}), we are very sorry..."`
    - If we **have this meal in the menu** and also, we **have all needed products** to make it, print the following message:
      `"Your order ({meal}) will be completed in the next 30 minutes and will cost you {the current price of the meal}."`
- You also **need to remove all used products** from those in stock and **add the price** of the meal to the **total budget**.

Examples

Sample Input

```
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']));
```

Output

```
Successfully loaded 10 Banana
Successfully loaded 20 Banana
Successfully loaded 50 Strawberries
Successfully loaded 10 Yogurt
There was not enough money to load 500 Yogurt
Successfully loaded 5 Honey
```

Sample Input

```
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));
```

Output

```
Great idea! Now with the frozenYogurt we have 1 meals on the menu, other ideas?
Great idea! Now with the Pizza we have 2 meals on the menu, other ideas?
```

Sample Input

```
console.log(kitchen.showTheMenu());
```

Output

```
frozenYogurt - $ 9.99
Pizza - $ 15.55
```

What to submit?

You are only required to submit the **Kitchen class**. No need to include the codes from the example above.

Class Signature: class **Kitchen**