# Exercises: Classes and Their Members

Problems for exercises and homework for the [“JavaScript Advanced” course @ SoftUni](https://softuni.bg/courses/javascript-advanced). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/340/>.

## Data Class

Write a JS class that holds data about an HTTP request. It has the following properties:

* method (String)
* uri (String)
* version (String)
* message (String)
* response (String)
* fulfilled (Boolean)

The first four properties (**method**, **uri**, **version**, **message**) are set trough the **constructor**, in the listed order. The **response** property is initialized to undefined and the **fulfilled** property is initially set to false.

### Input / Output

The constructor of your class will receive valid parameters. There is no output.

Submit the class definition as is, **without** wrapping it in any function.

### Examples

|  |  |
| --- | --- |
| Sample Input | Resulting object |
| let myData = new Request('GET', 'http://google.com', 'HTTP/1.1', '') | { method: 'GET',  uri: 'http://google.com',  version: 'HTTP/1.1',  message: '',  response: undefined,  fulfilled: false } |

### Hints

Using ES6 syntax, a class can be defined similar to a function, using the class keyword:



At this point, the class can already be **instantiated**, but it won’t hold anything useful, since it doesn’t have a constructor. A **constructor** is a function that initializes the object’s context and attaches values to it. It is defined with the keyword constructor inside the body of the class definition and it follows the syntax of regular JS functions – it can take arguments and execute logic. Any variables we want to be attached to the **instance** must be prefixed with the this identifier:



The description mentions some of the properties need to be set via the constructor – this means the constructor must receive them as parameters. We modify it to take four named parameters that we then assign to the local variables:



Note the input parameters have the same names as the instance variables – this isn’t necessary, but it’s easier to read. There will be no name collision, because the this identifier tells the interpreter to look for a variable in a different context, so this.method is not the same as method.

## Unity

Rats are uniting.

Create a class, Rat, which holds the functionality to unite with other objects of the same type. Make it so that the object holds all of the other objects it has connected to.

The class should have a **name**, which is a **string**, and it should be **initialized with it**.

The class should also hold a function unite(otherRat), which unites the **first object** with the **given one**. An object should store all of the objects it has united to. The function should only add the object if it is an object of the class Rat. In any other case it should **do nothing**.

The class should also hold a function getRats() which returns all the rats it has united to, in a list.

Implement functionality for toString() function… which returns a string representation of the object and all of the objects its united with, each on a new line. On the first line put the object’s name and on the next several lines put the united objects’ names, each with a padding of “**##**”.

### Example

|  |
| --- |
| test.js |
| **let *test*** = **new** Rat(**"Pesho"**); ***console***.log(***test***.toString()); *//Pesho* ***console***.log(***test***.getRats()); *//[]* ***test***.unite(**new** Rat(**"Gosho"**)); ***test***.unite(**new** Rat(**"Sasho"**)); ***console***.log(***test***.getRats()); *//[ Rat { name: 'Gosho', unitedRats: [] }, // Rat { name: 'Sasho', unitedRats: [] } ]* ***console***.log(***test***.toString()); *// Pesho // ##Gosho // ##Sasho* |

### Hints

Submit your solution as a class representation only! No need for IIFEs or wrapping of classes.

## Length Limit

Create a class, Stringer, which holds **single string** and a **length** property. The class should be initialized with a **string**, and an **initial length.** The class should always keep the **initial state** of its **given** **string**.

Name the two properties innerString and innerLength.

There should also be functionality for increasing and decreasing the initial **length** property.  
Implement function increase(length) and decrease(length), which manipulate the length property with the **given value**.

The length property is **a numeric value** and should not fall below **0**. It should not throw any errors, but if an attempt to decrease it below 0 is done, it should be automatically set to **0**.

You should also implement functionality for toString() function, which returns the string, the object was initialized with. If the length of the string is greater than the **length property**, the string should be cut to from right to left, so that it has the **same length** as the **length property**, and you should add **3 dots** after it, if such **truncation** was **done**.

If the length property is **0**, just return **3 dots.**

### Examples

|  |
| --- |
| test.js |
| **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* |

### Hints

Store the initial string in a property, and do not change it. Upon calling the toString() function, truncate it to the **desired value** and return it.

Submit your solution as a class representation only! No need for IIFEs or wrapping of classes.