More Exercises: Prototypes and Inheritance

1. C# Console

Write **Mocha Unit tests** to verify the functionality of a JavaScript implementation of the C# Console **class**. If you've written some code in C#, you would know that you can format text using placeholders, an example would look like this:

```
Console.WriteLine("The sum of {0} and {1} is {2}", 3, 4, 7);
```

Here the first placeholder **`{0}`** is exchanged for the first parameter passed after the text template - **3**. The second placeholder **`{1}`** for the second parameter - **4** and so on.

You will be provided with a class **`Console`** which has similar functionality to the C# one. The **`Console`** should have a static method writeLine which supports the following:

- **`writeLine(string)`** - if only a single argument is passed and it is a string, the function should simply return it.
- **`writeLine(object)`** - if only a single parameter is passed and it is an object - return the **JSON** representation of the object.
- **`writeLine(templateString, parameters)`** - It should support the following:
  - If multiple arguments are passed, but the first is not a string - throw a **`TypeError`**.
  - If the number of **`parameters`** does not correspond to the number of placeholders in the template string - throw a **`RangeError`**.
  - If the placeholders have indexes not withing the **`parameters`** range (for instance we have a placeholder **`{13}`** and only 5 params) throw a **`RangeError`**.
  - If multiple arguments are passed and the first is a string, find all placeholders from the string and **exchange** them with the supplied **`parameters`**.

Any cases not mentioned above, do not need to be checked.

Constraints

- All arguments in the **`writeLine(templateString, parameters)`** will be **strings**.
- There will never be two placeholders with the same number.

JS Code

To ease you in the process, you are provided with an implementation which meets all of the specification requirements for the **`Console`** object:

specialConsole.js

```
class Console {

  static get placeholder() {
    return /{\d+}/g;
  }

  static writeLine() {
    let message = arguments[0];
    if (arguments.length === 1) {
      if (typeof (message) === 'object') {
        message = JSON.stringify(message);
        return message;
      }
      else if (typeof(message) === 'string') {
        return message;
      }
    }
    else {
      if (typeof (message) !== 'string') {
        throw new TypeError("No string format given!");
      }
      else {
        let tokens = message.match(this.placeholder).sort(function (a, b) {
          a = Number(a.substring(1, a.length - 1));
          b = Number(b.substring(1, b.length - 1));
          return a - b;
        });
        if (tokens.length !== (arguments.length - 1)) {
          throw new RangeError("Incorrect amount of parameters given!");
        }
        else {
          for (let i = 0; i < tokens.length; i++) {
            let number = Number(tokens[i].substring(1, tokens[i].length - 1));
            if (number !== i) {
              throw new RangeError("Incorrect placeholders given!");
            }
            else {
              message = message.replace(tokens[i], arguments[i + 1])
            }
          }
          return message;
        }
      }
    }
  }
};
```

Your tests will be supplied a class named **"Console"** which contains the above-mentioned logic, all test cases you write should reference this variable. Submit in the Judge your code containing Mocha tests testing the above functionality.

2. Computer

You need to implement the class hierarchy for a computer business, here are the classes you should create and support:

- **`Keyboard`** class that contains:
  - **`manufacturer`** - string property for the name of the manufacturer
  - **`responseTime`** - number property for the response time of the Keyboard
- **`Monitor`** class that contains:
  - **`manufacturer`** - string property for the name of the manufacturer
  - **`width`** - number property for the width of the screen
  - **`height`** - number property for the height of the screen
- **`Battery`** class that contains:
  - **`manufacturer`** - string property for the name of the manufacturer
  - **`expectedLife`** - number property for the expected years of life of the battery
- **`Computer`** - **abstract** class that contains:
  - **`manufacturer`** - string property for the name of the manufacturer
  - **`processorSpeed`** - a number property containing the speed of the processor in GHz
  - **`ram`** - a number property containing the RAM of the computer in Gigabytes
  - **`hardDiskSpace`** - a number property containing the hard disk space in Terabytes
- **`Laptop`** - class **extending** the **`Computer`** class that contains:
  - **`weight`** - a number property containing the weight of the Laptop in Kilograms
  - **`color`** - a string property containing the color of the Laptop
  - **`battery`** - an instance of the **`Battery`** class containing the laptop's battery. There should be a **`getter`** and a **`setter`** for the property and validation that the passed in argument is actually an instance of the Battery class.
- **`Desktop`** - concrete class **extending** the **`Computer`** class that contains:
  - **`keyboard`** - an instance of the **`Keyboard`** class containing the Desktop PC's Keyboard. There should be a **`getter`** and a **`setter`** for the property and validation that the passed in argument is actually an instance of the Keyboard class.
  - **`monitor`** - an instance of the **`Monitor`** class containing the Desktop PC's Monitor. There should be a **`getter`** and a **`setter`** for the property and validation that the passed in argument is an instance of the **`Monitor`** class.

Attempting to instantiate an abstract class should throw an **`Error`**, attempting to pass an object that is not of the expected instance (ex. an object that is not an instance of Battery to the laptop as a battery) should throw a **`TypeError`**.

Example

computer.js

```
function createComputerHierarchy() {
    //TODO: implement all the classes, with their properties

    return {
        Battery,
        Keyboard,
        Monitor,
        Computer,
        Laptop,
        Desktop
    }
}
```

You are asked to submit **ONLY the function** that returns an object containing the above-mentioned classes.

Bonus:

In order to achieve a better code reuse, it's a good idea to have a base abstract class containing common information - check the classes, what common characteristics do they share that can be grouped in a common base class.

3. Mixins

Using the classes from the last task, write two mixins (functions which attach some functionality to passed in classes' prototypes) to extend their functionality. You need to support the following mixins:

- **`computerQualityMixin(classToExtend)`** - a function that attaches the following functions to the prototype of the passed in class.
    - **`getQuality()`** - returns a number equal to the computer's **`(processorSpeed + RAM + hardDiskSpace) / 3`**
    - **`isFast()`** - if **`processorSpeed > (ram / 4)`** returns **`true`**, otherwise **`false`**
    - **`isRoomy()`** - if **`hardDiskSpace > Math.floor(ram * processorSpeed)`** returns **`true`**, otherwise **`false`**
- **`styleMixin(classToExtend)`** - a function that attaches the following functions to the prototype of the passed in class:
    - **`isFullSet()`** - if the computer's **`manufacturer`**, **`keyboard's manufacturer`** and **`monitor's manufacturer`** all have the same name returns **`true`**, otherwise **`false`**.
    - **`isClassy()`** - if the computer battery's expected life is **3** years or **more** and the computer's color is either **`"Silver"`** or **`"Black"`** and the computer's weight is **less** than **3** kilograms returns **`true`**, otherwise returns **`false`**.

Examples

mixins.js

```
function createMixins() {
    // TODO: Create the mixins - computerQualityMixin and styleMixin, with all of their embedded functions
    return {
        computerQualityMixin,
        styleMixin
    }
}
```

You are asked to submit **ONLY the function** that returns an object containing the above mentioned mixins.