UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO
SCC0219 - Introduction to Web Development

Exercise 5: JavaScript II Classes

1. Expand the `BankAccount` class, presented in the class video, to include a method for adding interest to the account and a client object (from the class `Client`) with the client's name and address. Use the get/set operators when needed. With these new classes, write a program, using `alert()`, `prompt()` or `confirm()`, to (you can read a code to define the function):
    a. Create an account for a client.
    b. Deposit money.
    c. Deposit interest earnings (ex add 10% to balance: `account.interest(10)`).
    d. Withdraw money.
2. Create a class called `Shape` that has the type and perimeter read-only properties. Create a `Triangle` class that extends `Shape`. Objects from the `Triangle` class should have three properties a, b, and c representing the lengths of the sides of a triangle. Create a Square class from the Shape class adding a property side for the side length. Test your implementation with the following code:

```
const t = new Triangle(1, 2, 3);
console.log(t.type); // Triangle
console.log(t.perimeter); // 6
const q = new Square(2);
console.log(q.type); // Square
console.log(q.perimeter); // 8
q.perimeter = 9; // Error
```

3. A circular buffer is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end. A circular buffer first starts empty and of some predefined length. For example, this is a 5-element buffer:
   `[ ][ ][ ][ ][ ]`
   Assume that a 1 is written into the middle of the buffer (the exact starting location does not matter in a circular buffer):
   `[ ][ ][ ][1][ ]`
   Then assume that two more elements are added — 2 & 3 — which get appended after the 1:
   `[3][ ][ ][1][2]`
   If two elements are then removed from the buffer, the oldest values inside the buffer are removed. The two elements removed, in this case, are 1 & 2, leaving the buffer with just a 3:

`[3][ ][ ][ ][ ]`

Create a class CircularBuffer that implements this buffer with the following methods:

- Constructor with the size of the buffer.
- `toString()` returns a string representing the buffer, example: `Buffer = [3], [5]`
- `put(x)` to put an element in the current position. Returns the buffer. If the buffer is full, it writes over the oldest value.
- `pop()` to get and delete the oldest element. Throws an error if the buffer is empty.
- `size` read-only property to return the buffer size.
- `free` read-only property to return the buffer empty space.

Testing:

```
const buffer = new CircularBuffer(5);
console.log(buffer.size);  // 5
try { buffer.size = 9;} catch(e){} // Error
buffer.put(8).put('a').put(3);
// console.log may not call toString in some browsers
console.log(buffer.toString()); // Buffer = [8], ['a'], [3]
console.log(buffer.pop() + 'free: ' + buffer.free); // 8 free: 3
console.log(buffer.toString()); // Buffer = ['a'], [3]
buffer.put(4).put(5).put(6).put(7);
console.log(buffer.toString()); // Buffer = [3], [4], [5], [6],
[7]
buffer.pop();buffer.pop();buffer.pop();
buffer.pop();buffer.pop();
buffer.pop(); // Error: Circular buffer is empty.
```

Extra:

1. Regular expressions can be very useful in JavaScript to test user input, but they are sometimes tricky to create. Write a JavaScript function to check, using a regular expression, whether a given value is a valid USP URL or not. A valid USP URL is a valid URL that belongs to the `usp.br` domain. Tip: use the https://regex101.com/ site to test your regular expression.

Good luck!