GitHub link to source code: https://github.com/bb245/Design-atterns/tree/master/DecoratorPattern

Design pattern

In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Some of the original work on patterns (and wikis by the way) came from Portland Design Repository – http://c2.com/ppr. The concept of patterns was conceived of by a real, building-type architect. It describes common spaces, their functions and attributes in buildings and their relations to other patterns and spaces.

In software pattern descriptions follow the format listed below. It describes the problem the pattern addresses, the solution and what some of the pros and cons of using the pattern are. The first book on patterns is Gamma et al.

The pattern structure

Pattern name
- Pattern name and classification
- AKA
- The problem

Intent
- Motivation
- Applicability

The solution
- Structure
- Participants
- Collaboration
- Implementation – can be ignored, pseudo code at best
- Known uses
- Related patterns

The consequences
- Consequences

GitHub link to source code: https://github.com/bb245/Design-atterns/tree/master/DecoratorPattern

Why design patterns

There are three main reasons why you would want to use design patterns in JavaScript are:

Maintainability:
Design patterns help to keep your modules more loosely coupled. This makes it easier to refactor your code and swap out different modules. It also makes it easier to work in large teams and to collaborate with other programmers.

Communication:
Design patterns provide a common vocabulary for dealing with different types of objects. They give programmers shorthand for describing how their systems work. Instead of long explanations, you can just say, "It uses the factory pattern." The fact that a particular pattern has a name means you can discuss it at a high level, without having to get into the details.

Performance:
Some of the patterns we cover in this book are optimization patterns. They can drastically improve the speed at which your program runs and reduce the amount of code you need to transmit to the client. The flyweight and proxy patterns are the most important examples of this.

When it comes to patterns, the Gang of Four (or GoF) patterns are generally considered the foundation for all other patterns. They are categorized in three groups: Creational, Structural, and Behavioral. Here you will find information on these important patterns. You can find an overview of all GoF patterns in this doFactory design patterns section doFactory. Design Patterns are language-agnostic. That is, you can implement a software design pattern in any programming language. However, some of these patterns are more applicable to JavaScript than others.

Types of design patterns

Creational Design Patterns - focus on the different ways of creating objects and classes.

Example: Builder Pattern – allows you to construct objects without the need to explicitly create them; you only have to specify the type and the content of the object that you are creating. See the jQuery code snippet below:

```
var sampleDiv = $('<div id="IS683Div">This is a div</div>');
//a DOM node is now being referenced by the IS683Div jQuery object
var dummyText = $('<p/>');
//the HTML ParagraphElement is now being referenced by the dummyText
JQuery object
var testInput = $('< testInput />');
```

The result of the 3 lines of code above is a jQuery object referencing a DOM element. By using the builder pattern, you will be able to focus on the type and content of the object that you're creating instead of spending time to explicitly create it.

Structural Design Patterns – focus on the different ways of managing relationships between objects for an application to be architected in a scalable way. These patterns ensure that a change or update in one part of an application is independent and does not affect other parts.

Example: Facade Pattern – provides browser incompatibility solutions while providing users with a simple interface to use. The ready() method is one of the more popular JavaScript library method implementing the Facade pattern.

```
$(document).ready(function() {
//code goes here });
```

Other methods implemented using Facade include animate() and css().

Behavioral Design Patterns – All object-oriented software systems will involve communication between objects. The purpose of Behavioral Patterns is to provide different ways of organizing this communication.

Example: Observer Pattern – works with a subject and an observer

- subject has several observers interested in its lifecycle
- a subject updates or does something interesting, a notification is sent to all of its observers
- If an observer does not find the update interesting or is no longer interested with future updates, it can then remove itself from the list of observers

Here are 3 methods, supported by most JavaScript libraries, to further describe the pattern:

publish(data): called every time the subject has a notification to be sent

subscribe(observer): called every time the subject wants to add another observer to its current list of observers

unsubscribe(observer): called every time the subject wants to remove an observer from its current list of observers

```
var observer = $({});
$.subscribe = observer.on.bind(observer);
$.unsubscribe = observer.off.bind(observer);
$.publish = observer.trigger.bind(observer);
//usage
document.on( 'messageReceived', function(pinged)){
//code for performing actions and firing an event goes here
$.publish('messageShow', pinged);
 });
//subsribing to the event created above
 $.subscribe('messageShow', function() {
//code to display the message
//publish the action after they're  shown
 $.publish('messageDisplayed');
});
$.subscribe('messageDisplayed', function() {
});
```

The great thing about using design patterns is that other developers have already applied them successfully in the past, plus they are event-based so JavaScript can easily adopt them.

Combination of two design patterns – Decorator and constructor patterns

The program built will calculate the simple interest based on the country and additional interest will be given to senior citizens. Before jumping into the code here are some basics:

GitHub link to source code: https://github.com/bb245/Design-atterns/tree/master/DecoratorPattern

If you invest $10,000 for 3 years at 5% simple annual interest

*Simple interest = p * i * n = 10,000 * .05 * 3 = 1,500*

As said earlier, in addition to the above we decide to pay additional interest to senior citizens and this additional interest rates are assumed to be different for different countries [USA and CANADA (two countries considered in the program)].Then we can achieve this by utilizing the power of decorator pattern coupled with constructor pattern.

The implementation starts with a constructor and prototype methods:

```
function simpleInterest(principal,interest, period) {
                this.principal = principal;
                this.interest = interest;
                this.period = period;
            }

            simpleInterest.prototype.getSimpleInterest = function() {
                return  this.principal * this.interest * this.period ;

 };
```

Then, decorator object will be implemented as properties of constructor

```
simpleInterest.decorators = {};
```

Decorator object implements the customized getSimpleInterest ()

```
simpleInterest.decorators.fedSimpleInterestCalcForSenrCitizen = {
   getSimpleInterest: function() {
           var interest = this._super.getSimpleInterest();
           interest +=  interest*(3.0/100);
           return interest;
            }
};
```

In the same lines we can implement other decorators.

Finally, let's see the "magic" method called decorate() that ties all the pieces together.

```
simpleInterest.prototype.decorate = function (decorator) {
                var Fun = function () {},
                overrides = this.constructor.decorators[decorator],
                i,
                newobj;

                // Create prototype chain
                Fun.prototype = this;
                newobj = new Fun();
                newobj._super = Fun.prototype;

                // Mixin properties/methods of our decorator
                // Overriding the ones from our prototype
                for (i in overrides) {
                    if (overrides.hasOwnProperty(i)) {
                        newobj[i] = overrides[i];
                    }
                }

                return newobj;
            };
```

The newly decorated object *newobj* will inherit the object we have so far (either the original, or the one after the last decorator has been added), which is the object *this*. To do the inheritance part, let's use the temporary ***constructor pattern.***

Thus, using decorator and constructor design patterns the program permits us to add features to an object without needing to subclass it. Instead we "decorate" (wrap) it with another object with the same interface that has the one feature we're adding. That is to say, we can add country specific `SimpleInterestCalcForSenrCitizen` whenever we want to reward additional interest to senior citizens we appropriately include as below:

```
var simpleInterestforFed = new simpleInterest(10000,5,3);
```

```
  simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalc');
  simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalcForSenrCitizen');
simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterest');
```

Let us consider the following cases:

case 0: Simple interest is calculated for a CAN senior citizen who can enjoy additional interest from both CAN and US over a Principal amount $ 10,000, invested for a duration of 3 years @ 5% simple interest p.a, lucky one ahh! Below is the input and expected output.

Input

```
var simpleInterestforFed = new simpleInterest(10000,5,3);
simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalcForSenrCitizen');
simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalcForSenrCitizen');
simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterestQubec');
```

Expected Output

```
Decorator-constructor pattern example calculating simple Interest for a CAN
senior citizen

CAN$ 1606.80
```

Case I: Simple interest calculated for US senior citizen for a Principal amount $ 10,000, invested for duration of 3 years @ 5% simple interest p.a and an additional privilege interest of 3% p.a

Input

```
var simpleInterestforFed = new simpleInterest(10000,5,3);
simpleInterestforFed = simpleInterestforFed.decorate('fedSimpleInterestCalc');
simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalcForSenrCitizen');
simpleInterestforFed = simpleInterestforFed.decorate('finalSimpleInterest');
```

Expected Output

```
Decorator-constructor pattern example calculating simple Interest for a USA
senior citizen

USA$ 1545.00
```

GitHub link to source code: https://github.com/bb245/Design-atterns/tree/master/DecoratorPattern

Case II: Simple interest calculated for US senior citizen for a Principal amount $ 10,000, invested for duration of 3 years @ 5% simple interest p.a

```
Input
var simpleInterestforFed = new simpleInterest(10000,5,3);
simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalc');
simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterest');
Expected Output

Decorator-constructor pattern example calculating simple Interest for a USA
citizen

USA$ 1500.00
```

Case III: Simple interest calculated for CAN senior citizen for a Principal amount $ 10,000, invested for duration of 3 years @ 5% simple interest p.a and an additional privilege interest of 4% p.a

```
Input
var simpleInterestforFed = new simpleInterest(10000,5,3);
simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalc');
simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalcForSenrCitizen');
simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterestQubec');
Expected Output

Decorator-constructor pattern example calculating simple Interest for a CAN
senior citizen

CAN$ 1560.00
```

Case IV: Simple interest calculated for CAN senior citizen for a Principal amount $ 10,000, invested for duration of 3 years @ 7.5% simple interest p.a

```
Input
var simpleInterestforFed = new simpleInterest(10000,7.5,3);
simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalc');
simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterestQubec');
Expected Output

Decorator-constructor pattern example calculating simple Interest for a CAN
senior citizen

CAN$ 2250.00
```

The program as whole:

Index.html

GitHub link to source code: https://github.com/bb245/Design-atterns/tree/master/DecoratorPattern

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Decorators and Constructor patterns</title>
    <meta name="IS 683 >Decorators and Constructor design patterns
illustration" content="">
    <meta name="Bala" content="">
    <script type="text/javascript" src="decodemo.js" >

    </script>
```

Decodemo.js

```
"use strict";
            function simpleInterest(principal,interest, period) {
                this.principal = principal;
                this.interest = interest;
                this.period = period;
            }

            simpleInterest.prototype.getSimpleInterest = function() {
                return  this.principal * this.interest * this.period ;

            };

            simpleInterest.decorators = {};
            // for country CAN
            simpleInterest.decorators.qubecSimpleInterestCalc = {
                getSimpleInterest: function() {
                    var interest = this._super.getSimpleInterest();
                    interest = interest/100;
                    return interest;
                }
            };
```

```
simpleInterest.decorators.qubecSimpleInterestCalcForSenrCitizen = {
                getSimpleInterest: function() {

                    var interest = this._super.getSimpleInterest();
                    interest +=  interest*(4.0/100);
                    return interest;
                }
            };

            simpleInterest.decorators.finalSimpleInterestQubec = {
                getSimpleInterest: function() {
                    return "CAN$ " +
this._super.getSimpleInterest().toFixed(2);
                }
            };

            // for country USA
            simpleInterest.decorators.fedSimpleInterestCalc = {
                getSimpleInterest: function() {
                    var interest = this._super.getSimpleInterest();
                    interest = interest/100;
                    return interest;
                }
            };


simpleInterest.decorators.fedSimpleInterestCalcForSenrCitizen = {
                getSimpleInterest: function() {
                    var interest = this._super.getSimpleInterest();
                    interest +=  interest*(3.0/100);
                    return interest;
                }
            };
```

```
            simpleInterest.decorators.finalSimpleInterest = {
                getSimpleInterest: function() {
                    return "USA$ " +
this._super.getSimpleInterest().toFixed(2);
                }
            };


            simpleInterest.prototype.decorate = function (decorator) {
                var Fun = function () {},
                overrides = this.constructor.decorators[decorator],
                i,
                newobj;

                // Create prototype chain
                Fun.prototype = this;
                newobj = new Fun();
                newobj._super = Fun.prototype;

                // Mixin properties/methods of our decorator
                // Overriding the ones from our prototype
                for (i in overrides) {
                    if (overrides.hasOwnProperty(i)) {
                        newobj[i] = overrides[i];
                    }
                }

                return newobj;
            };
            document.write( "<em> Interest rate values used to compute
simple Interests for countries are only for demonstration
purposes.</em>");
            //USA senior citizen
```

```
            document.write ("<pre>function
simpleInterest(principal,interest, period)</pre>")

            //CAN citizen
            document.write( "<pre style=\"color:brown\">Case 0</pre>");
            document.write ("<b style=\"color:green\";><em>Simple
interest calculated for a CAN senior citizen who can enjoy additional
interest from both CAN and US over a Principal amount $ 10,000,
invested for duration of 3 years @ 5% simple interest p.a lucky one
ahh!</em></b>");

            document.write( "<pre style=\"color:blue\">Input</pre>");
            document.writeln ( "<pre>var simpleInterestforFed = new
simpleInterest(10000,5,3);</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalcForSenrCitizen');
</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalcForSenrCitizen');
</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterestQubec');</pre>");

            document.write( "<pre style=\"color:blue\">Output</pre>");

            //Simple interest calculator for CAN citizen for a
Principal = $ 10,000, invested for duration of 3 years  @ 5% simple
interest p.a
            var simpleInterestforFed = new simpleInterest(10000,5,3);

            simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalc');
            simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalcForSenrCitizen');
```

```
            simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalcForSenrCitizen');
            simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterestQubec');

            document.write( "<p style=\"color:maroon\"> Decorator-
constructor pattern example calculating simple Interest for a CAN
senior citizen</p>" + simpleInterestforFed.getSimpleInterest());



            document.write( "<pre style=\"color:brown\">Case I</pre>");
            document.write ("<b style=\"color:green\";><em>Simple
interest calculated for US senior citizen for a Principal amount $
10,000, invested for duration of 3 years @ 5% simple interest p.a and
an additional privilege interest of 3% p.a</em></b>");

            document.write( "<pre style=\"color:blue\">Input</pre>");
            document.writeln ( "<pre>var simpleInterestforFed = new
simpleInterest(10000,5,3);</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalc');</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalcForSenrCitizen');</
pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterest');</pre>");

            document.write( "<pre style=\"color:blue\">Output</pre>");

            //Simple interest calculator for US senior citizen for a
Principal = $ 10,000, invested for duration of 3 years  @ 5% simple
interest p.a
            var simpleInterestforFed = new simpleInterest(10000,5,3);
```

```
            simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalc');
            simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalcForSenrCitizen');
            simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterest');

            document.write( "<p style=\"color:maroon\"> Decorator-
constructor pattern example calculating simple Interest for a USA
senior citizen</p>" + simpleInterestforFed.getSimpleInterest());

            //USA  citizen
            document.write( "<pre style=\"color:brown\">Case
II</pre>");
            document.write ("<b style=\"color:green\";><em>Simple
interest calculated for US senior citizen for a Principal amount $
10,000, invested for duration of 3 years @ 5% simple interest
p.a</em></b>");

            document.write( "<pre style=\"color:blue\">Input</pre>");
            document.writeln ( "<pre>var simpleInterestforFed = new
simpleInterest(10000,5,3);</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalc');</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterest');</pre>");

            document.write( "<pre style=\"color:blue\">Output</pre>");

            //Simple interest calculator for US senior citizen for a
Principal = $ 10,000, invested for duration of 3 years  @ 5% simple
interest p.a
            var simpleInterestforFed = new simpleInterest(10000,5,3);
```

```
            simpleInterestforFed =
simpleInterestforFed.decorate('fedSimpleInterestCalc');
            simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterest');

            document.write( "<p style=\"color:maroon\"> Decorator-
constructor pattern example calculating simple Interest for a USA
citizen</p>" + simpleInterestforFed.getSimpleInterest());

            //CAN senior citizen
            document.write( "<pre style=\"color:brown\">Case
III</pre>");
            document.write ("<b style=\"color:green\";><em>Simple
interest calculated for CAN senior citizen for a Principal amount $
10,000, invested for duration of 3 years @ 5% simple interest p.a and
an additional privilege interest of 4% p.a</em></b>");

            document.write( "<pre style=\"color:blue\">Input</pre>");
            document.writeln ( "<pre>var simpleInterestforFed = new
simpleInterest(10000,5,3);</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalc');</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalcForSenrCitizen');
</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterestQubec');</pre>");

            document.write( "<pre style=\"color:blue\">Output</pre>");

            //Simple interest calculator for CAN senior citizen for a
Principal = $ 10,000, invested for duration of 3 years  @ 5% simple
interest p.a
            var simpleInterestforFed = new simpleInterest(10000,5,3);
```

```
            simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalc');
            simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalcForSenrCitizen');
            simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterestQubec');

            document.write( "<p style=\"color:maroon\"> Decorator-
constructor pattern example calculating simple Interest for a CAN
senior citizen</p>" + simpleInterestforFed.getSimpleInterest());


            //CAN citizen
            document.write( "<pre style=\"color:brown\">Case
IV</pre>");
            document.write ("<b style=\"color:green\";><em>Simple
interest calculated for CAN senior citizen for a Principal amount $
10,000, invested for duration of 3 years @ 7.5% simple interest
p.a</em></b>");

            document.write( "<pre style=\"color:blue\">Input</pre>");
            document.writeln ( "<pre>var simpleInterestforFed = new
simpleInterest(10000,7.5,3);</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalc');</pre>");
            document.writeln ( "<pre>simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterestQubec');</pre>");
            document.write( "<pre style=\"color:blue\">Output</pre>");

//Simple interest calculator for CAN citizen for a Principal = $
10,000, invested for duration of 3 years  @ 5% simple interest p.a
```

GitHub link to source code: https://github.com/bb245/Design-atterns/tree/master/DecoratorPattern

```
var simpleInterestforFed = new simpleInterest(10000,7.5,3);
simpleInterestforFed =
simpleInterestforFed.decorate('qubecSimpleInterestCalc');
simpleInterestforFed =
simpleInterestforFed.decorate('finalSimpleInterestQubec');

document.write( "<p style=\"color:maroon\"> Decorator-constructor
pattern example calculating simple Interest for a CAN senior
citizen</p>" + simpleInterestforFed.getSimpleInterest());
```

Please scroll to the next page to see the Output of the above program

Interest rate values used to compute simple Interests for countries are only for demonstration purposes.

```
function simpleInterest(principal,interest, period)
```

Case 0

Simple interest calculated for a CAN senior citizen who can enjoy additional interest from both CAN and US over a Principal amount $ 10,000, invested for duration of 3 years @ 5% simple interest p.a lucky one ahh!

Input

```
var simpleInterestforFed = new simpleInterest(10000,5,3);

simpleInterestforFed = simpleInterestforFed.decorate('qubecSimpleInterestCalcForSenrCitizen');

simpleInterestforFed = simpleInterestforFed.decorate('fedSimpleInterestCalcForSenrCitizen');

simpleInterestforFed = simpleInterestforFed.decorate('finalSimpleInterestQubec');
```

Output

Decorator-constructor pattern example calculating simple Interest for a CAN senior citizen

CAN$ 1606.80

Case I

Simple interest calculated for US senior citizen for a Principal amount $ 10,000, invested for duration of 3 years @ 5% simple interest p.a and an additional privilege interest of 3% p.a

Input

```
var simpleInterestforFed = new simpleInterest(10000,5,3);

simpleInterestforFed = simpleInterestforFed.decorate('fedSimpleInterestCalc');

simpleInterestforFed = simpleInterestforFed.decorate('fedSimpleInterestCalcForSenrCitizen');

simpleInterestforFed = simpleInterestforFed.decorate('finalSimpleInterest');
```

Output

Decorator-constructor pattern example calculating simple Interest for a USA senior citizen

USA$ 1545.00

Case II

Simple interest calculated for US senior citizen for a Principal amount $ 10,000, invested for duration of 3 years @ 5% simple interest p.a

Input

```
var simpleInterestforFed = new simpleInterest(10000,5,3);

simpleInterestforFed = simpleInterestforFed.decorate('fedSimpleInterestCalc');

simpleInterestforFed = simpleInterestforFed.decorate('finalSimpleInterest');
```

Output

Decorator-constructor pattern example calculating simple Interest for a USA citizen

USA$ 1500.00

Case III

Simple interest calculated for CAN senior citizen for a Principal amount $ 10,000, invested for duration of 3 years @ 5% simple interest p.a and an additional privilege interest of 4% p.a

Input

```
var simpleInterestforFed = new simpleInterest(10000,5,3);

simpleInterestforFed = simpleInterestforFed.decorate('qubecSimpleInterestCalc');

simpleInterestforFed = simpleInterestforFed.decorate('qubecSimpleInterestCalcForSenrCitizen');

simpleInterestforFed = simpleInterestforFed.decorate('finalSimpleInterestQubec');
```

Output

Decorator-constructor pattern example calculating simple Interest for a CAN senior citizen

CAN$ 1560.00

Case IV

Simple interest calculated for CAN senior citizen for a Principal amount $ 10,000, invested for duration of 3 years @ 7.5% simple interest p.a

Input

```
var simpleInterestforFed = new simpleInterest(10000,7.5,3);

simpleInterestforFed = simpleInterestforFed.decorate('qubecSimpleInterestCalc');

simpleInterestforFed = simpleInterestforFed.decorate('finalSimpleInterestQubec');
```

Output

Decorator-constructor pattern example calculating simple Interest for a CAN senior citizen

CAN$ 2250.00

GitHub link to source code: https://github.com/bb245/Design-patterns/tree/master/DecoratorPattern

References

Learning JavaScript Design Patterns A book by Addy Osmani
Javascript Patterns by Stoyan Stefanov
Pro JavaScript™ Design Patterns by Ross Harmes and Dustin Diaz
Hiddenwebgenius.com
Sourcemaking.com

End of document