EXAMEN: Sistemes i Tecnologies Web Juny 2018

Niu: Nom:

Permutació A

☞ NodeJS: Fancy Decorators

Si entregueu via moixero.uab.cat, ompliu el següent requadre. Altrament, ompliu la resta de l'examen.

Entrega Electrònica

The SHA1 checksum of the received file is:

.....

Time stamp is:

.....

Guardeu-vos una copia de l'arxiu que entregueu.

⇒ L'entrega no electrònica puntua sobre 7 / 10. ⇐

Exercici

Versió abreviada de l'enunciat

A – Fes la funció **debugDecorator** tal que executant el codi...

```
function f(p) { console.log("hola " + p) }
var f2 = debugDecorator(f)

f2(4)
f2(5)
```

...s'imprimeixin els seguents missatges per consola.

```
function f called 1 times (p = 4)
hola 4
function f called 2 times (p = 5)
hola 5
```

És a dir, **f2** imprimeix informació de debug abans de cridar **f**.

B – Fes la classe **RandomDelayDecorator** tal que executant el codi...

```
function f(p) { console.log("hola " + p) }

rdd = new RandomDelayDecorator()
rdd.setMaxDelay(500)

var f2 = rdd.decorate(f)

f(4)
f2(5)
```

...s'imprimeixin els seguents missatges per consola.

```
hola 4
hola 5
```

Atenció: el missatge **hola 4** s'ha d'imprimir inmediatament i el missatge **hola 5** s'ha d'imprimir en diferit fent servir **setTimeout()**. El diferiment ha de ser d'entre 500/2 i 500 millisegons després de la crida a **f2**.

C – Fes una server function que imprimeixi ≪Loading.... done!≫ de forma progressiva com la que es mostra a continuació, però fent que NodeJS esperi entre 5 i 10 mil·lisegons entre punt i punt.

```
function serverFunction(request, response) {
        //Sending HTTP Response headers
        response.writeHead(200,
                {'Content-Type': 'text/plain; charset=utf-8'});
        response.write('Loading')

        for (var count = 0; count < 1024; count++) {
                response.write('.')
        }

        response.write(' done!')
        response.end()
}
```

Cal que facis servir **RandomDelayDecorator**.

Especificacions

debugDecorator (2 punts)

La idea aquí és fer un embolcall al voltant d'una funció, per tal de que s'imprimeixi informació de debug al cridar la funció. Això és fa canviant la funció per una altra, que primer imprimeix informació de debug i després crida a la funció original.

Síntaxi:

```
debugDecorator(f)
```

Paràmetres:

f – una funció d'un sol paràmetre **p**, i.e., **f(p)**.

Valor Retornat

f2 – una funció d'un sol paràmetre **p**, i.e., **f2(p)**.

Funcionalitat

- **f2** primer fa un **console.log** i després crida **f**.
- El **console.log** ha d'imprimir:

```
'function ' + f.name + ' called ' + times + ' times (p = ' + p + ')'
```

- **f** s'ha de cridar amb el mateix paràmetre p que rep **f2**.
- **f2** ha de retornar el resultat de **f(p)**.

Exemple d'us:

```
var f2 = debugDecorator(function(x) { console.log("hola!") }); f2(1)
```

Resultat:

```
function called 1 times (p = 1)
```

RandomDelayDecorator (4 punts)

Es tracta d'un cas semblant a l'anterior. La classe **`RandomDelayDecorator`** permet fer un embolcall al voltant d'una funció, però en aquest cas el que es fa és diferir l'execució de la funció.

Métodes:

`setMaxDelay`

Síntaxi

`setMaxDelay(milliseconds)`

Paràmetres

`milliseconds` – Un valor enter.

Funcionalitat

Setter de la propietat **`maxDelay`**.

`decorate`

Síntaxi

`decorate(f)`

Paràmetres

`f` – una funció d'un sol paràmetre **`p`**, i.e., **`f(p)`**.

Valor Retornat

`f2` – una funció d'un sol paràmetre **`p`**, i.e., **`f2(p)`**.

Funcionalitat

– **`f2`** programa una futura execució de **`f`** amb **`setTimeout()`** i retorna inmediatament.

– **`f`** s'ha de cridar amb el mateix paràmetre p que rep **`f2`**.

– **`f2`** no ha de retornar res.

Indicacions:
– La funció **`setTimeout(f, m, p1, p2, ...)`** crida la funció **`f`** després de que hagin passat **`m`** milisegons de temps. Els arguments **`p1`**, **`p2`**, **`...`** són els paràmetres que se li passaran a **`f`** quan es cridi. Nota: l'espera de **`setTimeout`** és asíncrona. **`setTimeout`** no s'espera a que passi el temps, sinó que retorna inmediatament de la mateixa manera que ho fa **`fs.readFile`**.
– Podeu fer servir **`Math.random()`** per generar un valor aleatori entre 0 i 1. També podeu fer servir aquest codi:

```
m = (0.5 + 0.5 * Math.random()) * milliseconds
```

serverFunction (4 punts)

Cal fer servir **`RandomDelayDecorator`** per fer que la funció que es mostra a continuació esperi entre 5 i 10 milisegons entre punt i punt.

```
function serverFunction(request, response) {
        //Sending HTTP Response headers
        response.writeHead(200,
                {'Content-Type': 'text/plain; charset=utf-8'});
        response.write('Loading')

        for (var count = 0; count < 1024; count++) {
                response.write('.')
        }

        response.write(' done!')
        response.end()
}
```

Indicació sobre com resoldre l'exercici:

1. Oblidat un moment dels entre 5 i 10 milisegons.
2. Transforma el **`for`** en una funció recursiva, que a cada nivell de recursió imprimeixi un punt. I.e.,

   "**`f(x)`** = si **`x < 1024`**, imprimeix un punt i crida **`f(x + 1)`**, sinó tanca la conexió."

3. Canvia la funció **`f`** per una altra resultant de **`decorate()`**.

Cal provar-ho amb Firefox.

Test i Entrega

Ompliu el vostre codi a l'arxiu **`fancydecorator.js`**.
1. Executeu el vostre servidor de NodeJS: nodejs fancydecorator.js
2. Aquest arxiu incorpora un conjunt de testos que han de resultar en:

```
Server running at http://localhost:8081/
-------------------------------------------------
Tests for debugDecorator
-------------------------------------------------
function f1 called 1 times (p = 3)
  f1 function called with x = 3
  Is return value -6? true

function f1 called 2 times (p = 4)
  f1 function called with x = 4
  Is return value -8? true

function f1 called 3 times (p = undefined)
  f1 function called with x = undefined
  Is return value NaN? true

function f2 called 1 times (p = 1)
  f2 function called with x = 1
  Is return value undefined? true

function f2 called 2 times (p = undefined)
  f2 function called with x = undefined
  Is return value undefined? true

-------------------------------------------------
Tests for RandomDelayDecorator
-------------------------------------------------
launching test4 ok? true
launching test4 ok? true
launching test5 ok? true
function  called 1 times (p = at [0, 0])
launching test6 ok? true
launching test4 ok? true
function  called 2 times (p = at [250, 500])
launching test6 ok? true
waiting...

  f3 function called with at [0, 0]
function f4 called 1 times (p = at [0, 0])
  f4 function called with at [0, 0]
  f3 function called with at [0, 0)
  f3 function called with at [100, 200)
  f3 function called with at [100, 200)
function f4 called 2 times (p = at [250, 500])
  f4 function called with at [250, 500]
```

3. Obriu un navegador i accediu al servidor amb la URL: http://localhost:8081

Entregueu l'arxiu **`fancydecorator.js`** via moixero si us ha funcionat tot, o copieu el codi que tingueu als requadres disponibles al final d'aquest document.

Resposta

Ompliu els següents espais. Recordeu que la mida no dona cap indicació del nombre de línies de codi que us calen.

```
http = require('http')

// TODO: implement function debugDecorator here

var debugDecorator = function (f) {









}

// TODO: implement class RandomDelayDecorator here

var RandomDelayDecorator = function() {









}
```

```
// TODO: implement server function

/*

Should do something like this, but waiting between 5 and 10 milliseconds
between printing each dot.

function serverFunction(request, response) {
        //Sending HTTP Response headers
        response.writeHead(200, {'Content-Type': 'text/plain; charset=utf-8'});
        response.write('Loading')

        for (var count = 0; count < 1024; count++) {
                response.write('.')
        }

        response.write(' done!')
        response.end()
}
*/

function serverFunction(request, response) {









}

http.createServer(serverFunction).listen(8081);

console.log('Server running at http://localhost:8081/');
```

```
// Tests:

console.log("-----------------------------------------------------")
console.log("Tests for debugDecorator")
console.log("-----------------------------------------------------")

var test1 = debugDecorator(function f1(x) {
        console.log("  f1 function called with x = " + x); return -2 * x })

console.log("  Is return value -6? " + (test1(3) === -6) + "\n")
console.log("  Is return value -8? " + (test1(4) === -8) + "\n")
console.log("  Is return value NaN? " + (isNaN(test1())) + "\n")

var test2 = debugDecorator(function f2(x) {
        console.log("  f2 function called with x = " + x); })

console.log("  Is return value undefined? " + (test2(1) === undefined) + "\n")
console.log("  Is return value undefined? " + (test2() === undefined) + "\n")

console.log("-----------------------------------------------------")
console.log("Tests for RandomDelayDecorator")
console.log("-----------------------------------------------------")

var rdd1 = new RandomDelayDecorator()
var rdd2 = new RandomDelayDecorator()

var f3 = function f3(x) {
        console.log("  f3 function called with " + x); return "f3: " + x };
var f4 = function f4(x) {
        console.log("  f4 function called with " + x); return "f4: " + x };

rdd1.setMaxDelay(200)

var test4 = rdd1.decorate(f3)
var test5 = rdd2.decorate(f3)
var test6 = debugDecorator(rdd2.decorate(debugDecorator(f4)))

console.log("launching test4 ok? " + (test4("at [100, 200)") === undefined))
console.log("launching test4 ok? " + (test4("at [100, 200)") === undefined))
console.log("launching test5 ok? " + (test5("at [0, 0]") === undefined))
console.log("launching test6 ok? " + (test6("at [0, 0]") === undefined))

rdd1.setMaxDelay(0)
rdd2.setMaxDelay(500)

console.log("launching test4 ok? " + (test4("at [0, 0)") === undefined))
console.log("launching test6 ok? " + (test6("at [250, 500]") === undefined))

console.log("waiting...\n")
```