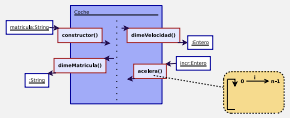
Grado Tecnologías Interactivas

Programación 2



Práctica 5

UNIVERSIDAD POLITECNICA DE VALENCIA

Escola Politècnica Superior de Gandia

DSIC

Departament de Sistemes Informàtics i Computació

Práctica 5

¡ Atención !

▷ Se recuerda que las prácticas deben prepararse antes de acudir al aula informática, anotando en el enunciado las dudas que se tengan.

▷ Los diseños y algortimos que se piden en esta práctica deben escribirse en la libreta de apuntes para poder ser revisados.

▷ Utiliza *git* en cada ejercicio haciendo *commit* cada vez que consigas un "hito".

▷ La realización de las prácticas es un trabajo individual y original. En caso de plagio se excluirá al alumno de la asignatura. Por tanto, es preferible presentar el trabajo realizado por uno mismo aunque éste tenga errores.

1

Clases

Escribir clases en JavaScript resulta muy sencillo. Como ejemplo, estudiemos cómo se implementa la conocida clase Punto en este lenguaje.

```
class Punto {
 // - - - - - - - - - - - - - - -
 // x:R, y:R ->
 //             f() ->
 // - - - - - - - - - - - - - - -
 constructor( x, y ) {
  this.x = x
  this.y = y
 }
 // - - - - - - - - - - - - - - -
 //      f() <-
 // R  <-
 // - - - - - - - - - - - - - - -
 getX () {
  return this.x
 }
 // - - - - - - - - - - - - - - -
 //      f() <-
 // R  <-
 // - - - - - - - - - - - - - - -
 getY () {
  return this.y
 }
 // - - - - - - - - - - - - - - -
 // Punto ->
 //         f() <-
 //    R <-
 // - - - - - - - - - - - - - - -
 distancia( otro ) {
  var dx = this.x - otro.x
  var dy = this.y - otro.y
  return Math.sqrt( (dx*dx) + (dy*dy) )
 }
} // class

// ----------------
// main
// ----------------
var p1 = new Punto( 0, 0 )
var p2 = new Punto( 3, 4 )

var d = p1.distancia( p2 )

console.log( d )
```

Ejercicio.

1. Prueba el anterior código.
2. Busca la implementación de la clase Punto en C++ y compara su código con el anterior.

2 Organización del código. Módulos

Resulta evidente que el código fuente de cualquier programa no trivial debe estar divido en distintos ficheros para

- ser más claro y legible; y tener una buena organización del código, acorde con su diseño.
- poder ser desarrollado en equipo.

node.js dispone de un mecanismo "export-require" de cuyo uso vamos a ver un primer ejemplo: el fichero bib.js contiene dos funciones que van a ser utilizadas desde main.js.

```
// ---------------
// bib.js
// ---------------

// ---------------
// R -> porDos() -> R
// ---------------
module.exports.porDos = function ( a ) {
 return a * 2
} // ()

// ---------------
// R -> porTres() -> R
// ---------------
module.exports.porTres = function ( a ) {
 return a * 3
} // ()
```

```
// ---------------
// main.js
// ---------------

// ---------------
// requires
// ---------------
var bib = require( "./bib.js" )

// ---------------
// main ()
// ---------------
var a = bib.porDos( 8 )
console.log( a )

console.log( bib.porTres( 3 ) )
```

Como se ve, el fichero que exporta funciones, añade dichas funciones como propiedades del objeto `module.exports`. El fichero que importa las funciones, con `require`, obtiene dicho objeto y lo guarda en una variable para su posterior uso. Si lo que queremos exportar es una clase, podremos hacerlo como en el siguiente ejemplo.

```
// -----------------
// Punto.js
// -----------------
module.exports = class Punto {
    // - - - - - - - - - - - - - - - -
    // - - - - - - - - - - - - - - - -
 constructor( x, y ) {
  this.x = x
  this.y = y
 }
    // - - - - - - - - - - - - - - - -
    // - - - - - - - - - - - - - - - -
 getX () {
  return this.x
 }
    // - - - - - - - - - - - - - - - -
    // - - - - - - - - - - - - - - - -
 getY () {
  return this.y
 }
} // class
```

```
// -----------------
// main.js
// -----------------

// -----------------
// requires
// -----------------
const Punto = require( "./Punto.js" )

// -----------------
// main ()
// -----------------
var p1 = new Punto( 3, 4 )

console.log( p1.getX() )
```

Ejercicio.

1. Prueba el anterior código.
2. Diseña y añade un nuevo método a la clase Punto (en el fichero Punto.js) que devuelva un nuevo punto el cuál sea la diferencia de otros dos a y b, siendo $nuevo = ((a.x - b.x), (a.y - b.y))$. (Nota: al implementarlo, evidentemente, uno de los puntos será: this.)

3 npm

Cuando se desarrollan programas de cierta envergadura es inevitable utilizar bibliotecas desarrolladas por terceros. Obtener una biblioteca manualmente supone ir a la página web que la ofrece, buscar la versión que nos interesa, descargar dicha versión en algún lugar de nuestro ordenador y configurar nuestro programa para que encuentre la biblioteca descargada. Y este trabajo hay que hacerlo para cada bilioteca que usemos, y cada vez que una biblioteca sea actualizada.

Este trabajo es tedioso y propenso a errores. Para automatizarlo hay varias opciones. Una opción muy general es usar git con los comandos clone o pull. Aunque posible, no suele ser la solución ideal porque normalmente obtenemos todo el código fuente de la biblioteca, que no queremos modificar sino sólo utilizar su versión compilada.

La solución más correcta es utilizar un "gestor de paquetes". *node.js* tiene uno: npm (*node package manager*), que vamos a aprender a utilizar.

Para utilizar npm conviene escribir un pequeño fichero de configuración llamado package.json. Como indica su extensión, el contenido de este fichero es estrictamente JSON. El fichero package.json se puede crear con este comando

```
npm init
```

que formulará una serie de cuestiones para confeccionar el fichero.

Sin embargo, suele ser más práctico tener un fichero package.json como plantilla y copiarlo y adpatarlo cada vez que empezamos un programa.

Ejercicio.

Supongamos que queremos escribir un pequeño servidor web utilizando la biblioteca *express*. Sigue los siguientes pasos

1. En nuevo directorio, copia el siguiente contenido en package.json

```
{
 "name": "prueba_servidor",
 "version": "1.0.0",
 "description": "prueba de servidor express",
 "main": "main.js",
 "author": "Jordi",
 "repository": "ninguno de momento",
 "license": "ISC",
 "dependencies": {
  "express": "*"
 },
 "scripts": {
  "ayuda": "echo ejecuta esto: npm run main",
  "main": "node main.js"
 }
}
```

2. Ejecuta

```
npm install
```

para instalar las dependencias. Las dependencias son las bibliotecas que queremos utilizar, que en este caso es sólo express; como hemos indicado en package.json:

```
"dependencies": {
 "express": "*"
},
```

3. Copia el siguiente fichero en el directorio de trabajo. (Léelo y trata de entender qué hace. En el futuro se explicará en detalle).

```
// ----------------------------
// main.js
// ----------------------------
// requires
const express = require( "express" )

// ----------------------------
// main()
// ----------------------------
// creo un servidor
const servidor = express()

// cuando llegue GET /hola
servidor.get("/hola", function( peticion, respuesta ) {
 respuesta.send( "Hola a todos")
})

// cuando llegue GET /adios
servidor.get("/adios", function( peticion, respuesta ) {
 respuesta.send( "Hasta pronto")
})
```

```
// arranco el servicio en el puerto 8080
servidor.listen( 8080, function() {
 console.log( "Escuchando en el puerto 8080")
 console.log( "Conéctate a localhost:8080/hola" )
})
```

4. Ejecuta.

```
npm run ayuda
```

Localiza en pacakge.json dónde está definido qué hace este comando.

5. Ejecuta.

```
npm run main
```

Conectáte a las URLs http://localhost:8080/hola y http://localhost:8080/adios en tu navegador. Relaciónalo con el programa main.js.

4

Test automáticos. mocha

Conocemos ya la importancia de realizar test automáticos a nuestros programas, o a partes de ellos, como medida de calidad para que otros puedan usar con garantías el código que les proporcionamos. Si siempre son necesarios, en JavaScript son ineludibles porque la mayoría de errores se detectan exclusivamente al ejecutar el programa.

Ejercicio.

Vamos a aprender utilizar una bilbioteca llamada *mocha* para aplicar test de forma automática a nuestro código. Sigue los siguientes pasos.

1. Dentro de un nuevo directorio de trabajo, crea el subdirectorio `test`, y copia los siguientes ficheros para que al final todo quede de la siguiente forma:

```
|-- Punto.js
|-- node_modules
|-- package.json
|-- test
    |-- mainTest1.js
    |-- mainTest2.js
```

Fichero Punto.js

```
// -----------------
// Punto.js
// -----------------
module.exports = class Punto {
    // - - - - - - - - - - - - - - - -
    // - - - - - - - - - - - - - - - -
 constructor( x, y ) {
  this.x = x
  this.y = y
 }
    // - - - - - - - - - - - - - - - -
    // - - - - - - - - - - - - - - - -
 getX () {
  return this.x
 }
    // - - - - - - - - - - - - - - - -
    // - - - - - - - - - - - - - - - -
 getY () {
  return this.y
 }
    // - - - - - - - - - - - - - - - -
    // - - - - - - - - - - - - - - - -
 distancia( otro ) {
  var dx = this.x-otro.x
  var dy = this.y-otro.y
  return Math.sqrt( dx*dx + dy*dy )
 }
} // class
```

Fichero package.json: lee este fichero con suma atención. Ejecuta npm install tras copiarlo para que se descarguen las dependencias en node_modules

```
{
 "name": "clase_punto",
 "version": "1.0.0",
 "description": "desarrollo y pruebas de la clase
Punto",
 "main": "Punto.js",
 "author": "Jordi",
 "repository": "ninguno de momento",
 "license": "ISC",
 "dependencies": {
  "mocha": "*"
 },
 "scripts": {
  "test": "node ./node_modules/mocha/bin/mocha",
  "ayuda": "echo ejecuta esto: npm test"
 }
}
```

Fichero test/mainTest1.js: lee atentamente este fichero para entender cómo se escriben los test.

```
const Punto = require('../Punto.js')
var assert = require ('assert')
// --------------
// main ()
// --------------
// descripción los test que hacemos aquí
describe( "Prueba constructor y getters ",
    function () {
     // ---------------------
     before( function( hecho ) {
      console.log("esto ocurre antes de los it()")
      hecho() // llamo a esta funcion para seguir
     })

     // ---------------------
     // it(): test concreto
     it( "pruebo getY()", function( hecho ){
      var p1 = new Punto( 3, 4 )

      // Esta es la comprobación:
      // compruebo que getY() da 4
      assert.equal( p1.getY(), 4 )
      hecho()
     }) // it

     // ---------------------
     // it(): test concreto
     it( "pruebo getX()", function( hecho ){
      var p1 = new Punto( 3, 4 )

      // Vamos a simular que esto es un test
      // asíncrono poniendo un timeout.
      setTimeout( function() {
       // Esta es la comprobación:
       // compruebo que getX() da 3
       assert.equal( p1.getX(), 3 )
       hecho() /* llamo a hecho para inidicar
          * que este test ha terminado,
          * y así vamos al siguiente it()  */
      }, 500)
     }) // it

     // ---------------------
     after( function() {
      console.log("esto ocurre después de los it()")
     })
    } ) // describe
```

Fichero test/mainTest2.js: lee atentamente este fichero para entender cómo se escriben los test.

```
const Punto = require('../Punto.js')
var assert = require ('assert')

// --------------
// main ()
// --------------
// descripción los test que hacemos aquí
describe( "Prueba de distancia()",
    function () {
     var p1 = new Punto( 0, 0 )
     var p2 = new Punto( 3, 4 )

     // ---------------------
     // it(): test concreto
     it( "la distancia de p1 a p2 es 5", function(
hecho ){
      assert.equal( p1.distancia(p2), 5 )
      hecho()
     }) // it

     // ---------------------
     // it(): test concreto
     it( "la distancia de p2 a p1 es 5", function(
hecho ){
      assert.equal( p1.distancia(p2), 5 )
      hecho()
     }) // it

     // ---------------------
     // it(): test concreto
     it( "la distancia de p1 a p1 es 0", function(
hecho ){
      assert.equal( p1.distancia(p1), 0 )
      hecho()
     }) // it
    } ) // describe
```

2. Ejecuta en el directorio donde está package.json:

```
npm test
```

y comprueba que aparece lo siguiente.

```
Prueba constructor y getters
esto ocurre antes de los it()
  ■ pruebo getY()
  ■ pruebo getX() (503ms)
esto ocurre después de los it()

Prueba de distancia()
  ■ la distancia de p1 a p2 es 5
  ■ la distancia de p2 a p1 es 5
  ■ la distancia de p1 a p1 es 0

5 passing (522ms)
```

Ejercicio.

Añade el método diferencia() implementado en una sección anterior a tu clase Punto y escribe en un nuevo fichero test/mainTest3.js un test para probar el método diferencia().

16 abril 2020