Entrega 2 - Eventos

Versión: 11 de Febrero de 2019

Objetivo

El objetivo de esta práctica es aprender cómo funciona la programación asíncrona basada en eventos implementada por el modulo **events** y la clase **EventEmitter**, así como el uso de **callbacks**, **clases** y **módulos** de Node.js.

Enunciado de la práctica

Esta práctica proporciona un programa de ejemplo está formado por 4 ficheros: **habitacion.js**, **climatizador.js**, **termostato.js** y **practica2.js**. Este programa simula la habitación de una vivienda que tiene instalados un climatizador y un termostato que controla su temperatura. El programa de ejemplo usa del módulo **events** de Node.js para gestionar los eventos.

Tal y como se indica en la sección ”Realización y prueba de la práctica”, estos ficheros se descargan con el proyecto de prueba: https://github.com/practicas-ging/CORE19-02_events.

¡Cuidado! El código dado funciona en UNIX/Linux. Para ejecutar en Windows hay que aplicar los cambios que se indican en las instrucciones sobre compatibilidad UNIX-Windows para node.js.

En esta práctica el alumno deberá realizar dos tareas:

- Tarea 1: Ampliar el programa de ejemplo añadiendo un programador. El programador permitirá configurar que temperatura ideal se desea tener en la habitación en todo momento.
- Tarea 2: Sustituir el módulo **events** de Node.js por uno equivalente implementado por el alumno, de forma que el funcionamiento del programa de ejemplo no se vea afectado.

Descripción del programa de ejemplo

El programa de ejemplo simula la habitación de una vivienda que tiene instalado un climatizador para enfriarla o calentarla. La habitación también tiene un termostato en el que se puede configurar una temperatura ideal, y que controla el funcionamiento del climatizador.

El programa de ejemplo está formado por cuatro ficheros: **habitacion.js**, **climatizador.js**, **termostato.js** y **practica2**.js.

Para ejecutar el programa de ejemplo se debe invocar el comando:

```
$ node practica2.js
```

*El fichero **habitación.js***

El fichero **habitacion.js** es un módulo que exporta la clase **Habitacion**.

Cada instancia de la clase **Habitacion** simula una habitación de una vivienda, y mantiene el valor de su temperatura ambiente.

Además, cada instancia de **Habitacion** tiene un temporizador que modifica la temperatura aleatoriamente en +/- 1ºC cada 10 segundos. Esta variación aleatoria de la temperatura simula los cambios de estación, o cualquier actividad externa que afecte a la temperatura de la habitación.

```
// Clase habitacion.
// Su temperatura cambia aleatoriamente. (Simula el invierno o el verano)
class Habitacion {

    constructor() {

        // Temperatura actual de la habitacion:
        this.temperatura = 20.0;

        // Cada 10 segundos sube o baja aleatoriamente la temperatura
        // hasta +/- un grado:
        setInterval(() => {
            this.temperatura += Math.random() * 2 - 1,
            console.log(`Cambio aleatorio a ${this.temperatura.toFixed(1)}ºC`);
        }, 10000);
    }
}

exports = module.exports = Habitacion;
```

*El fichero **climatizador.js***

Este fichero es un módulo exporta la clase **Climatizador**.

Cada instancia de la clase **Climatizador** simula un aparato climatizador instalado en una

```
// Cuanto subimos o bajamos la temperatura
const DELTA = 0.1;


// Clase Climatizador.
// Modifica la temperatura de una habitacion.
// Metodos:
//    enfriar
//    calentar
class Climatizador {

       constructor(habitacion) {

              this.habitacion = habitacion;
       }

       enfriar() {
              console.log('Enfriando.')
        this.habitacion.temperatura -= DELTA;
       }

       calentar() {
              console.log('Calentando.')
        this.habitacion.temperatura += DELTA;
       }
}

exports = module.exports = Climatizador;
```

habitación, y que permite aumentar y disminuir la temperatura de la habitación. Toma aire de la habitación y lo expulsa un poco más caliente o un poco más frio.

El constructor de la clase **Climatizador** toma como parámetro un objeto **Habitación** que representa la habitación donde está instalado el climatizador.

Los objetos climatizador tienen dos métodos: **enfriar** y **calentar**. Se usan para disminuir o aumentar ligeramente la temperatura de la habitación cada vez que se invocan.

*El fichero **termostato.js***

Este módulo exporta la clase **Termostato**.

```
const EventEmitter = require('events');

// Diferencia de temperatura permitida entre la temperatura real y la ideal.
const MARGEN_ERROR = 0.3;

// Clase Termostato.
// Mira la temperatura de una habitacion y avisa si hace demasiado calor o frio.
// Tambien informa sobre la temperatura actual de la habitacion.
// Metodos:
//    indicarTemperaturaIdeal
//    encender
//    apagar
// Eventos:
//    tic
//    muchocalor
//    muchofrio
class Termostato  extends EventEmitter {

    constructor(habitacion) {
        super();

        this.habitacion = habitacion;

        // Temperatura ideal programada:
        this.temperaturaIdeal = 16;

        // para cancelar el temporizador setInterval:
        this.intervalId = null;
    }

    indicarTemperaturaIdeal(temperaturaIdeal) {
        this.temperaturaIdeal = temperaturaIdeal;
    }

    encender() {
        console.log('Encendiendo el termostato.');
        clearInterval(this.intervalId);
        this.intervalId = setInterval(() => {
            this.emit('tic', this.habitacion.temperatura);

            if (this.habitacion.temperatura > this.temperaturaIdeal+MARGEN_ERROR) {
                this.emit('muchocalor');
            } else if (this.habitacion.temperatura < this.temperaturaIdeal-MARGEN_ERROR) {
                this.emit('muchofrio');
            }
        }, 500);
    }

    apagar() {
        console.log('Apagando el termostato.');
        clearInterval(this.intervalId);
    }
}

exports = module.exports = Termostato;
```

Cada instancia de la clase **Termostato** simula un termostato instalado en una habitación, que puede configurarse con una temperatura ideal, y que emite eventos informando cuando la temperatura de la habitación es demasiado alta o baja respecto de la ideal.

También emite un evento periódico informando de la temperatura actual de la habitación.

La clase **Termostato** extiende a la clase **EventEmitter** para gestionar la emisión de eventos y sus escuchadores.

El constructor de la clase **Termostato** toma como parámetro un objeto **Habitación** que representa la habitación donde está instalado.

Cada instancia de **Termostato** tiene tres propiedades:

- La propiedad **habitacion** mantiene una referencia a la habitación donde está instalado el termostato.
- La propiedad **temperaturaIdeal** almacena el valor de la temperatura ideal que se ha configurado.
- La propiedad **intervalId** identifica un temporizador interno que periódicamente sondea la temperatura de la habitación y emite los eventos "**tic**", "**muchocalor**" y "**muchofrio**". La propiedad **intervalId** se necesita para cancelar el temporizador interno cuando se apaga o se vuelve a encender el termostato.

Cada instancia de **Termostato** tiene tres métodos:

- El método **indicarTemperaturaIdeal** se usa para configurar la temperatura ideal deseada en la habitación.
- El método **encender** se usa para encender el termostato, inicializando el temporizador interno que lanza los eventos.
- El método **apagar** se usa para apagar el temporizador, detenido el temporizador interno encargado de lanzar los eventos.

Las instancias de **Termostato** pueden emitir tres eventos:

- El evento "**muchocalor**" se emite cuando la temperatura actual de la habitación supera a la temperatura ideal en más grados que los indicados por la constante **MARGEN_ERROR**.
- El evento "**muchofrio**" se emite cuando la temperatura actual de la habitación es inferior a la temperatura ideal en más grados que los indicados por la constante **MARGEN_ERROR**.
- El evento "**tic**" se emite cada medio segundo, y le pasa a los escuchadores como parámetro el valor de la temperatura actual de la habitación.

*El fichero **practica2.js***

Este fichero es el programa principal.

Importa las clases **Habitacion**, **Climatizador** y **Termostato**, y crea una instancia de cada una de ellas.

La constante **dormitorio** apunta a una habitación que inicialmente está a 22ºC.

Las constantes **climatizador** y **termostato** apuntan al climatizador y al termostato instalados en el dormitorio anterior.

En el termostato se registran tres escuchadores. El primero escucha los eventos "**muchofrio**" para provocar que el climatizador caliente. El segundo escucha los eventos "**muchocalor**" para que provocar que el climatizador enfríe. El último escucha los eventos "**tic**" para mostrar por la consola la temperatura actual del dormitorio.

Finalmente, se configura una temperatura ideal a 20ºC y se enciende el termostato.

```
const Habitacion = require('./habitacion');
const Climatizador = require('./climatizador');
const Termostato = require('./termostato');

// Creamos una habitacion:
const dormitorio = new Habitacion();
dormitorio.temperatura = 22;

// Creamos un climatizador para la habitacion:
const climatizador = new Climatizador(dormitorio);

// Creamos un Termostato que mira la temperatura de la habitacion:
const termostato = new Termostato(dormitorio);

// Configuramos el termostato para controlar la temperatura:
termostato.on('muchofrio', () => climatizador.calentar());
termostato.on('muchocalor', () => climatizador.enfriar());

// Mostar la temperatura periodicamente:
termostato.on('tic', (temp) => console.log(`${temp.toFixed(1)}ºC`));

// Configurar la temp ideal a 20 grados:
termostato.indicarTemperaturaIdeal(20);

// Encender el termostato:
termostato.encender();
```

Tarea 1

La primera tarea que debe realizar el alumno es ampliar el programa de ejemplo añadiendo un programador que permita configurar la temperatura que se desea tener en la habitación en todo momento.

Debe crearse un fichero llamado **programador.js** que implemente un módulo que exporte una clase llamada **Programador**.

El constructor de la clase **Programador** debe tomar como parámetro un objeto con la configuración de horas y temperaturas que se desea programar. El objeto con la configuración debe ser un array como el ilustrado en el ejemplo de la derecha. Cada elemento del array es un objeto con dos claves:

```
[
  { hora: "07:00",
    temperatura: 22
  },
  { hora: "08:30",
    temperatura: 18
  },
  { hora: "18:00",
    temperatura: 22
  },
  { hora: "23:00",
    temperatura: 20
  }
]
```

- **hora**: El valor asociado a esta clave es un string con la hora a la que debe aplicarse la programación de la nueva temperatura ideal. El formato de la hora debe ser "hh:mm".
- **temperatura**: un número con la nueva temperatura ideal a programar.

Con la configuración del ejemplo se programaría una temperatura de 22ºC todos los días a las 7 de la mañana, 18ºC todos los días a las 8:30, 22ºC todas las tardes a las 6, y 20ºC todos los días a las 11 de la noche.

Se recomienda usar el módulo **Later.js** (https://bunkat.github.io/later) para implementar el módulo **programador.js**. Recuerde instalarlo con npm install. El módulo **Later.js** permite planificar instantes de tiempo en los que ejecutar tareas. Por ejemplo, para escribir por consola la palabra "hola" todos los días a las 18:00, se podría usar el siguiente código:

```
// Importar modulo Later.js:
const later = require('later');

// Usar zona horaria local:
later.date.localTime();

// Crear planificación para todos los dias a las 18:00
const sched = later.parse.text('at 18:00');

// Crear un temporizador que escriba indefinidamente "hola"
// en los instantes planificados anteriormente:
later.setInterval(() => console.log('hola'), sched);
```

Las instancias de la clase **Programador** deben emitir un evento llamado "**ideal**" cada vez que sea necesario reprogramar la temperatura ideal, siguiendo las instrucciones de la configuración pasada en el constructor. El evento "**ideal**" se emitirá pasando como parámetro el valor de la temperatura ideal a programar en el termostato.

Para poder emitir eventos, la clase **Programador** debe extender a la clase **EventEmitter**.

Finalmente, en el programa principal **practica2.js** deben añadirse los siguientes cambios:

- Importar el módulo **programador.js** para crear un objeto **Programador**.
- Añadir las sentencias necesarias para que cuando el programador emita un evento "**ideal**", se ajuste la nueva temperatura ideal en el termostato.

Tarea 2

En esta tarea se pide al alumno que cree su propia implementación del módulo **events**, y que la integre en el programa de ejemplo, sustituyendo al módulo **events** proporcionado por Node.js.

La implementación pedida al alumno es una versión reducida de la proporcionada por Node.js. El alumno solo debe implementar los métodos **emit** y **on**.

Para que el módulo implementado por el alumno se integre fácilmente en el programa de ejemplo, deben cumplirse los siguientes requisitos:

- El módulo debe llamarse **events.js** y situarse junto a los demás ficheros javascripts.
- Para requerirlo desde **termostato.js** y desde **programador.js** deberá sustituir las llamadas **require("events")** por **require("./events")**.
- El módulo debe exportar una clase que se llame **EventEmitter**.
- Las instancias de **EventEmitter** deben tener una propiedad donde se deben guardar todos los escuchadores que se hayan registrado usando el método **on**. Se debe guardar el nombre de los eventos junto con la lista de escuchadores interesados en cada evento.
- Las instancias de **EventEmitter** deben tener el método **on**, para que los escuchadores se registren. Este método toma como parámetro el nombre de un evento, y el método a ejecutar cuando se emita ese evento.
- Las instancias de **EventEmitter** deben tener el método **emit**, que se emplea para emitir un evento. Este método toma como parámetros el nombre del evento a emitir, y los argumentos que hay que pasar a las funciones escuchadoras interesadas en ese evento.

Una vez sustituido el módulo events de Node.js por el desarrollado por el alumno, el comportamiento del programa de prueba no debe variar.

Realización y prueba de la práctica

Para comprobar que la práctica ha sido realizada correctamente hay que utilizar el validador de este repositorio

https://github.com/practicas-ging/CORE19-02_events

Recuerde que para utilizar el validador se debe tener node.js (y npm) (https://nodejs.org/es/) y Git instalados. El proyecto se descarga, instala y ejecuta en el ordenador local con estos comandos:

```
$                                ## El proyecto debe clonarse en el ordenador local
$ git clone https://github.com/practicas-ging/CORE19-02_events
$
$ cd CORE19-02_events            ## Entrar en el directorio de trabajo
$
$ npm install                    ## Instala el programa de test
$
$                            ##  ->  Incluir la solución en el esqueleto clonado
$
$ npm run checks                          ## Pasa los tests a los ficheros del proyecto
..................................... ## solicitado en el directorio de trabajo
.....................................
... (resultado de los tests)
$
```

El proyecto local descargado estará en el directorio **CORE19-02_events**. Este proyecto incluirá los ficheros **habitacion.js**, **climatizador.js**, **termostato.js** y **practica2.js** descritos en el enunciado. reutilizando dichos ficheros debe realizar las tareas 1 y 2 del enunciado.

¡Cuidado! El código dado funciona en UNIX/Linux. Para ejecutar en Windows habrá que aplicar los cambios que se indican en las instrucciones sobre compatibilidad UNIX-Windows para node.js.

Los tests pueden pasarse las veces que sea necesario y en cualquiera de las etapas de desarrollo. Si se pasan nada más descargar el proyecto, indicarán que no se ha realizado todavía nada. El programa de test incluye además un comando para generar el fichero ZIP una vez finalizada la práctica

```
$
$ npm run zip         ##  Comprime los ficheros del directorio en un fichero .zip
$
```

Este comando genera el fichero **CORE19-02_events_entregable.zip** con el directorio de la practica comprimido. Este fichero ZIP debe subirse a la plataforma para su evaluación.

Instrucciones para la Entrega.

La entrega debe realizarse subiendo a la plataforma un fichero ZIP con el contenido comprimido del directorio del proyecto **CORE19-02_events,** generado con el comando “npm run zip”. Este contiene los ficheros **habitacion.js**, **climatizador.js**, **termostato.js**, **practica2.js***,* ***package.json***, junto con los demás ficheros que se hayan creado.