Práctica patrones de diseño

Patrón modular

Author: Nacho Lereu Ramírez

El objetivo de esta práctica es aplicar un patrón de diseño modular a un diseño realizado por el analista para generar un módulo de calculo de ritmos de paso para el entrenamiento y carreras de atletas.

Para ello, deberéis de generar con el patrón modular un modulo ***calculator*** en un fichero independiente js que debe de contener dos submódulos: ***distanceConverter*** y ***timeConerter***.
Por cuestiones de notación cualquier variable, función , propiedad o método deberá seguir una notación ***camelCase***.

El submódulo ***timeConverter*** debe presentar los siguientes métodos:

```
/**      timeToSeconds
*
* @param  {object}
* @return {seconds}
* @description convert time object
* @example { hours: 1, minutes: 34, seconds: 23} in seconds.
*/

/**  secondsToTime
* @param  {Integer}
* @return {object}
* @description convert seconds in time object
* @example { hours: 1, minutes: 34, seconds: 23}.
*/
```

El submódulo ***distanceConverter*** debe presentar los siguientes métodos:

```
/**
 * kmToMeters
 * @param  {km}
 * @return {meters}
 * @description convert kilometers in meters.
 */
 /**
 * metersToKm
 * @param  {meters}
 * @return {km}
 * @description convert  meters in km
 */
 /**
 * milesToImperial
 * @param  {miles}
 * @return {object}
 * @description convert miles in imperial format
 * @example { miles: 1, yards: 343,  feets: 2}.
 *
 */
```

```
/**
* yardsToImperial
* @param  {yards}
* @return {object}
* @description convert yards in imperial format
* @example { miles: 1, yards: 343,  feets: 2}.
*/

/**
* imperialToMiles
* @param  {object}
* @return {miles}
* @description convert imperial format
* @example { miles: 1, yards: 343,  feets: 2} in miles.
*/

/**
* metersToMiles
* @param  {meters}
* @return {miles}
* @description convert  meters in miles
*/

/**
* milesToMeters
* @param  {miles}
* @return {meters}
* @description convert  miles in meters
*/
```

Para realizar las conversiones entre sistema métrico y el sistema imperial se dispone de la siguiente tabla:

Unit	Conversion
1 mile	1609.344 meters
1 mile	1760 yards
1 yard	3 feets

Finalmente el módulo ***calculator*** debe implementar los métodos:

```
 /**
  * [paceInKm calcula el ritmo de carrera por kilometro y milla a partir del tiempo en segundos realizado y la distancia recorrida en kilometros]
  * @param  {number} timeInSeconds
  * @param  {number} distanceInKm
  *  @return {[time,time]}    [devuelve un array con el ritmo por kilometro y ritmo por milla]
  */

 /**
  * [paceInMiles calcula el ritmo de carrera por kilometro y milla a partir del tiempo en segundos realizado y la distancia recorrida en millas]
  * @param  {number} timeInSeconds
  * @param  {number} distanceInMiles
  *  @return {[time,time]}    [devuelve un array con el ritmo por kilometro y ritmo por milla]
  */
```

```
 /**
   * [markFromPacePerKm: calcula la marca esperada al recorrer la
distancia en kilometros al ritmo de carrera por kilometro realizado]
   * @param  {time} pacePerKm
   * @param  {number} distanceInMeters
   * @return {time}   [devuelve el tiempo/marca esperado]
   */


/**
   * [markFromPacePerMile: calcula la marca esperada al recorrer la
distancia en millas al ritmo de carrera por milla realizado]
   * @param  {time} pacePerMile
   * @param  {imperial} distanceInImperial
   * @return {time}   [devuelve el tiempo/marca esperado]
   */
 /**
   * [tableTimeFromPacePerKm: calcula la marca esperada al recorrer la
distancia en metros al ritmo de carrera por kilometro cada
cutDistanceInMeters]
   * @param  {time} pacePerKm
   * @param  {number} distanceInMeters
   * @param  {number} cutDistanceInMeters
   * @return {time}    [devuelve un array de objetos con propiedades
distance=distanciaIntermedia mark=tiempo de paso en la distancia
intermedia]
   */

/**
   * [tableTimeFromPacePerMile: calcula la marca esperada al recorrer
la distancia en millas al ritmo de carrera por milla cada
cutDistanceInYards]
   * @param  {time} pacePerMile
   * @param  {number} distanceInMiles
   * @param  {number} cutDistanceInYards
   * @return {time}    [devuelve un array de objetos con propiedades
distance=distanciaIntermediaImperial mark=tiempo de paso en la
distancia intermedia]
   */
```

Una cuestión a destacar es lo que sucede cuando se hacen operaciones de división y multiplicación con javascript de forma consecutiva. Este tipo de operaciones son necesarias para expresar millas o kilómetros en el formato imperial. Observar este ejemplo:

(((2000/1760-1)*1760)-240)/3

Su resultado en javascript es 5.684341886080802e-14 cuando debería ser 0. Osea, 2000 millas son 1 milla 240 yardas y 5.684341886080802e-14 pies.

Si se hace Math.round(5.684341886080802e-14) se obtiene 0. Y puede parecer que la solución es suficiente con aplicar este método del objeto Math para calcular los pies. Sin embargo esto no es cierto.

Este hecho hace que en la implementación de los módulos deberíais crear una función que solvente este problema, si es que consideráis que es necesario.