Tema 4. Programación con funciones, arrays y objetos definidos por el usuario

Anexo. Objetos en JavaScript

1. Objetos predefinidos

Definición de objeto

En los temas anteriores hemos visto una serie de elementos, o mejor dicho, de clases de elementos a las que hemos llamado objetos. Un objeto es algo más que una variable, pues posee propiedades y métodos. Hemos visto que había objetos tanto en la estructura de la página o DOM como en el propio lenguaje javascript. Sin embargo no se ha explicado bien lo que es un objeto.

Un objeto es una estructura que tiene propiedades y métodos.

Las propiedades son como los adjetivos en el lenguaje, es decir que expresan una cualidad del objeto, así por ejemplo las instrucciones dadas en lenguaje CSS se consideran propiedades, ya que modifican el aspecto de la página. Otras propiedades pueden darnos información acerca del objeto, por ejemplo sobre su tamaño, valor, estado, etc. Los atributos de las etiquetas HTML son considerados también propiedades.

Los métodos son funcionalidades, es decir funciones o modos de operar asociados a un objeto. Se comportan igual que una función, es algo que se puede hacer con el objeto, es por eso que llevan siempre un paréntesis detrás, en el que se le pueden pasar varios valores para que opere con ellos (los parámetros o argumentos).

En javascript se accede a las propiedades y a los métodos de un objeto con el operador punto (.) tal como hemos visto en todos los ejemplos de lecciones anteriores.

Ejemplo:

```
Math.SQRT2
fecha.getFullYear()
```

Estructuras de objetos

Los objetos pueden formar estructuras jerárquicas, tal como sucede en la estructura DOM de la página. Aquí vemos como unos objetos tienen como propiedades a otros objetos, formando así una estructura en árbol.

Hay que diferenciar los objetos de la página o del DOM y los objetos de javascript, así los objetos de la página tienen estructura jerárquica, y dependen todos del objeto window. Dentro del documento dependen todos del objeto document. Es por eso por lo que la mayoría de las instrucciones que empleamos para acceder a la página empiezan por document.

Por otra parte el lenguaje javascript tiene también unos objetos predefinidos, algunos de los cuales los hemos visto en temas anteriores. Estos, mediante sus

propiedades y métodos, nos permiten realizar las tareas más habituales de una forma sencilla. Estos objetos tienen su jerarquía, y dependen todos del objeto Object, el cual veremos más adelante.

Otra característica de los objetos es la **herencia**. Esto consiste que si un objeto tiene unas propiedades y métodos, todos los objetos que jerárquicamente dependan de él, poseerán también (heredarán) las mismas propiedades y métodos. Esto es algo que hay que tener en cuenta en la estructura de la página, y que se suele tener muy en cuenta en la hoja de estilos o código CSS.

Objetos ya vistos en Javascript

En realidad, más que de objetos deberíamos hablar de clases de objetos, ya que un objeto en sí sería por ejemplo una fecha, y la clase de objeto a la que pertenece sería la clase Date.

Para crear un nuevo objeto de una clase ya definida, la forma habitual es escribir:

```
nuevoObjeto = new ClaseObjeto()
```

donde nuevoObjeto es el objeto nuevo que creamos. new es la palabra o operador que indica a javascript que estamos creando un nuevo objeto. ClaseObjetoes la clase a la que pertenecerá el nuevo objeto; dentro de los paréntesis, y dependiendo del tipo de objeto, podremos poner algunas características del objeto, por ejemplo en los arrays podemos poner los elementos que lo componen.

Sin embargo no es la única forma de crear un objeto, ya que dependiendo de la clase que se trate, puede haber otras formas de crearlo. Por ejemplo al asignarle un valor a una nueva variable, podemos estar creando un objeto tipo Número (Number) o Cadena de texto (String); también los arrays podemos crearlos de varias maneras.

Hasta ahora hemos visto en temas anteriores las siguientes clases de objetos:

- **Number**: Números
- **Math**: Operaciones con números
- **Date**: Fechas.
- **String**: Cadenas de texto.
- **Array**: Listas de variables.
- **Boolean**: valores booleanos o lógicos

Sin embargo nos faltan por ver algunas clases de objetos, los cuales aunque menos importantes desde el punto de vista de la programación debemos tenerlos en cuenta.

El objeto Function

Las funciones también se consideran objetos en Javascript, y forman la clase Function. Aunque la forma más habitual de declararlas es la que hemos visto: function miFuncion(); también lo podemos hacer mediante la forma general de construir objetos:

miFunción = new Function() { ... }

Tal como ocurre en la forma habitual de declararlas, dentro del paréntesis escribiremos, si hace falta, los parámetros que necesite la función.

No tiene propiedades ni métodos propios, sino los heredados del objeto Object que veremos más adelante.

El objeto Object

Como se ha apuntado anteriormente, el objeto Object es el que está en un nivel superior en la jerarquía, y del que derivan todos los demás objetos de javascript. Permite por lo tanto crear nuevas clases de objetos.

Sus métodos y propiedades son heredados por el resto de los objetos javascript, algunos de ellos ya los hemos visto antes y otros los explicaremos más detenidamente en páginas posteriores.

Los **métodos** son:

- *toString();* el cual devuelve siempre una cadena de texto con el nombre del objeto (por ejemplo transforma true en "true");
- *valueOf*(): Devuelve el valor primitivo del objeto (true/false, un número,...).

Y las **propiedades**:

- *constructor*: Devuelve la función que crea el objeto
  Ej: objeto.constructor devolverá function Object() {...}
- *prototype*: Permite añadir propiedades y métodos al objeto

Crear objetos.

Objetos nuevos.

Javascript nos permite crear nuestros propios objetos, con sus propiedades y métodos, los cuales también los podemos definir. Crear un nuevo objeto consiste básicamente en declararlo en una función, y aplicarle nuevas propiedades y métodos, veamos cómo se hace esto:

Ejemplos:

Definir objetos vacíos

```
alumno = new Object();
```

```
alumno = {};
```

```
alumno = new Object();
alumno.nombre="Juan";
alumno.curso = "2ºDAW";
alumno.materias = 5;
```

```
alumno = {nombre:"Juan",
curso:"2ºDAW", materias:5}
```

Crear mediante un constructor

Un constructor es una función que inicializa el objeto, para ello utilizamos una función normal, y en el paréntesis le pasamos algunos parámetros que nos servirán para definir las propiedades que tendrá el objeto.

Veámoslo con un ejemplo. Pongamos como ejemplo los alumnos que se matriculan en una academia, crearemos una clase Alumnos en la que cada alumno será un objeto. La función que la crea será la siguiente:

function Alumno(nombre,curso,numMaterias) { ... }

Propiedades

La función anterior es de momento una función normal a la que se le pasan unos parámetros, ahora vamos a convertir estos parámetros en propiedades del objetoAlumno para ello hacemos una referencia al parámetro dentro de la propia función con la palabra reservada this, de la siguiente manera:

this.alumno = nombre;

Esto lo escribimos dentro de la función, añadimos también las propiedades curso y numMaterias y la función quedará así:

```
function Alumno(nombre,curso,numMaterias) {
    this.alumno = nombre;
    this.curso = curso;
    this materias = numMaterias;
    }
```

Con esto ya tenemos una nueva clase de objetos, la clase Alumno, que de momento, solo tiene tres propiedades.

Al declarar las propiedades lo que ponemos después de this, será el nombre de la propiedad, y lo que ponemos después del igual es el parámetro que se convertirá en la propiedad, por lo que debe ser igual al que le pasamos en el paréntesis de la función.

Instanciar el objeto

Se llama instanciar un objeto a crear un objeto de una clase que ya está definida. Así cuando creamos un array mediante miArray = new Array() estamos instanciando un objeto de la clase array.

Con las clases de objetos nuevos creados mediante las funciones constructor, instanciaremos los objetos de la misma manera, así para crear un objeto concreto de la clase Alumno escribiremos:

alumno1 = new Alumno("Juan Pérez","tecnología",5)

Como ves se trata de crear un objeto de la misma manera que lo haríamos con otra case de objetos. Dentro del paréntesis debemos poner los parámetros que serán los valores de las propiedades del objeto, tal como está en la función constructor.

Ahora para ver las propiedades las llamamos mediante su nombre precedido de un punto:

nombre1 = alumno1.alumno

En el siguiente código mostramos la función constructor, un objeto instanciado, extraemos sus propiedades, y las mostramos en una ventana de alerta:

```
function Alumno(nombre,curso, numMaterias) {
    this.alumno = nombre;
    this.curso = curso;
    this.materias = numMaterias;
    }
alumno1 = new Alumno ("Juan Perez","tecnología",5);
nombre1 = "alumno: "+alumno1.alumno;
curso1 = " curso: "+alumno1.curso;
materias1 = " con "+alumno1.materias+" asignaturas.";
alert(nombre1 + curso1 + materias1);
```

También podríamos haber creado el objeto sin ningún parámetro en el paréntesis, y pasarle después los valores de las propiedades, de la siguiente manera:

alumno1 = new Alumno();

Después damos valor a las propiedades del objeto que acabamos de crear:

alumno1.alumno = "Juan Perez";
alumno1.curso = "informática"
alumno1.materias = 7

De la misma manera podemos cambiar el valor de cualquier propiedad, simplemente con asignarle un nuevo valor.

Métodos

Los métodos se crean de manera parecida, pero hay que recordar que los métodos son funciones, por lo tanto el método debe igualarse al nombre de una función que indique qué es lo que queremos hacer con el objeto en la función constructor pondremos lo siguiente para crear un método:

```
this.precioCurso = matricula
```

Donde precioCurso es el nombre que le daremos al método, y matricula es el nombre que le daremos a la función.

Observa que tanto el método como el nombre de la función se ponen aquí sin paréntesis, aunque luego al llamar al método sí debemos ponerlos.

El siguiente paso es crear la función a la que se refiere el método, en nuestro ejemplo, el método precioCurso() establecerá el precio en base al número de asignaturas, de manera que serán 100€ por asignatura, la función que controla este método es la siguiente:

```
function matricula() {
    precio = 100*this.materias
    return precio
    }
```

Esta función dice lo que debe hacer el método precioCurso(), la función puede hacer referencia a propiedades del objeto mediante la palabra reservada this, y debe tener siempre un valor de retorno con return que será el que es devuelto por el método.

Veamos el código del ejemplo anterior, tras añadir la función anterior, la referencia en la función constructor, y añadirlo al objeto instanciado, para poder verlo en una ventana de alerta:

```
function Alumno(nombre,curso,numMaterias) {
    this.alumno = nombre;
    this.curso = curso;
    this.materias = numMaterias;
    this.precioCurso = matricula;
    }
```

```
function matricula() {
        precio = 100*this.materias;
        return precio;
        }
alumno1 = new Alumno ("Juan Perez","tecnología",5);
nombre1 = "alumno: "+alumno1.alumno;
curso1 = " curso: "+alumno1.curso;
materias1 = " con "+alumno1.materias+" asignaturas.";
precio1 = " precio: " +alumno1.precioCurso()+ "€.";
alert(nombre1 + curso1 + materias1 + precio1 );
```

Métodos con argumentos

Podemos poner también un método que necesite algún dato para ser completado, el cual se le pasará a través del paréntesis, como un argumento, tal como se hace en las funciones. Para ello en la función a la que hace referencia el método introduciremos el argumento, el cual debemos pasarlo en el método.

Supongamos que en el ejemplo que tenemos, el precio final puede verse reducido en un tanto por ciento mediante una beca, para calcularlo, crearemos un nuevo método. En el parámetro de la función pasaremos el tanto por ciento de reducción del precio, el método lo creamos añadiendo a la función constructor la siguiente sentencia:

```
this.precioBeca = beca
```

y la función que controla este método será la función beca() la cual será la siguiente:

```
function beca(num){
        precio = 100*this.materias;
        descuento = precio*num/100;
        precioFinal = precio - descuento;
        return precioFinal;
        }
```

En esta función el argumento del paréntesis será el tanto por ciento de reducción en el precio, el cual deberemos pasarlo al aplicar el método precioBeca.

Veamos cómo queda el código del script tras crear este nuevo método y aplicarlo al objeto que tenemos de referencia.

```
function Alumno(nombre,curso,numMaterias) {
        this.alumno = nombre;
        this.curso = curso;
        this.materias = numMaterias;
        this.precioCurso = matricula;
        this.precioBeca = beca;
        }
function matricula() {
        precio = 100*this.materias;
        return precio
        }
function beca(num){
        precio = 100*this.materias;
        descuento = precio*num/100;
        precioFinal = precio - descuento;
        return precioFinal;
        }
alumno1 = new Alumno ("Juan Perez","tecnología",5);
nombre1 = "alumno: "+alumno1.alumno;
curso1 = " curso: "+alumno1.curso;
materias1 = " con "+alumno1.materias+" asignaturas.";
precio1 = " precio: " +alumno1.precioCurso()+ "€.";
beca1 = " Precio de becario: " +alumno1.precioBeca(15)+"€.";
alert(nombre1 + curso1 + materias1 + precio1 + beca1 );
```

La propiedad prototype I

Definición

La propiedad prototype es una propiedad del objeto Object, y por tanto es heredada por el resto de los objetos. Por lo tanto todos los objetos poseen la propiedad prototype.

Mediante esta propiedad podemos asignar nuevos métodos y propiedades a cualquier clase de objetos. Veamos cómo hacerlo:

Asignar nuevas propiedades

Podemos asignar nuevas propiedades a un objeto, tanto si este está predefinido en javascript, como si está definido por nosotros, con la propiedad prototype.

En el ejemplo que vamos a seguir aquí veremos cómo una función en blanco, es decir sin ningún código, se convierte en función constructor al asignarle propiedades y métodos mediante la propiedad prototype. Partimos de la siguiente función:

```
function Calcular() { }
```

Como ves, esta es una función sin código, pero ahora mediante la propiedad prototype convertiremos esta función en un constructor. Le asignaremos en principio dos propiedades. Para ello en primer lugar definiremos dos variables, que serán los valores por defecto de las dos propiedades:

```
a = 0; b = 0;
```

Después mediante la propiedad prototype asignamos estos valores como propiedades de la función anterior.

```
Calcular.prototype.n1 = a;
Calcular.prototype.n2 = b;
```

Donde n1 y n2 son los nombres de las propiedades de la función Calcular.

Para comprobar que la función Calcular es ahora un constructor, y tiene las propiedades n1 y n2, escribimos lo siguiente:

```
calculo = new Calcular();
alert(calculo.n1+ " , "+calculo.n2);
```

El resultado será una ventana que nos mostrará los valores de las propiedades n1 y n2 del objeto calculo; como no hemos modificado las propiedades, nos mostrará "0 , 0 ".

Sin embargo no hemos hecho todo este código sólo para mostrar una ventana de alerta que podríamos haber escrito de una forma mucho más fácil. Ahora lo que pretendemos, aparte de ver cómo funciona la propiedad prototype, es crear una página que calcule el área de un rectángulo, un triángulo, o un círculo, a partir de los números dados por el usuario.

De momento ya tenemos dos números, que son las propiedades del objeto calculo. Los cuales nos los dará el usuario mediante un formulario.

Ahora nos falta calcular las áreas de las figuras anteriores, para ello deberemos emplear funciones que las calculen es decir, métodos.

Asignar nuevos métodos

Asignar nuevos métodos a un objeto mediante la propiedad prototype se hace de la misma forma que asignar propiedades. La única diferencia es que en lugar de asignarle como valor por defecto una variable, se le asigna como valor una función.

Siguiendo con nuestro ejemplo, crearemos un método que calcule el área de un rectángulo, las propiedades n1 y n2 serán la longitud de los lados. Debemos crear primero la función que calcula el área, en ella la referencia a las propiedades anteriores se hace mediante la palabra reservada this. Esta es la función:

```
function areaRectangulo() {
    solucion = this.n1 * this.n2;
    return solucion;
}
```

El resultado debe ser devuelto mediante la palabra reservada return.

Ahora que tenemos la función la asignamos como método a la clase Calcular:

```
Calcular.prototype.areaRectangulo = areaRectangulo
```

Tenemos ahora un nuevo método llamado areaRectángulo(), que calcula el área de un rectángulo a partir de las propiedades n1 y n2 donde se escribirán la longitud de los lados.

Si queremos ampliar la página anterior, para calcular también las áreas del triángulo y el círculo, debemos seguir los mismos pasos para crear los métodos que las calculan, y después aplicarlos a un objeto concreto de la clase. En el código HTML debemos también añadir los botones que nos muestren el resultado de las operaciones. La página ya ampliada tendrá el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <meta charset="UTF-8">
    <title>propiedad prototype</title>
    <script type="text/javascript">

        a=0;b=0;

        function Calcular(){}

        Calcular.prototype.n1 = a; //propiedad número 1
        Calcular.prototype.n2 = b; //propiedad número 2
```

```
function areaRectangulo() { //función para crear método área de rectángulo
  solucion = this.n1 * this.n2;
  return solucion;
}

Calcular.prototype.areaRectangulo = areaRectangulo; //crear método a partir de la función.

function rectangulo(){ // función que muestra el resultado para rectángulo
  instanciar(); //llamada a la función que crea un objeto concreto.
  result = numeros.areaRectangulo(); //aplicar método
  operacion = "Area del rectángulo: "; //texto para pantalla
  escribir(); //función que escribe el resultado en pantalla.
}

function areaCirculo() { //función crear metodo area circulo
  solucion = Math.pow(this.n1,2)*Math.PI;
  return solucion;
}

Calcular.prototype.areaCirculo = areaCirculo //crear método a partir de la función.

function circulo() { //resultados para circulo
  instanciar();
  result = numeros.areaCirculo();
  operacion = "Area del círculo: ";
  escribir();
}

function areaTriangulo() { //función crear método área triángulo
  solucion = this.n1 * this.n2/2;
  return solucion;
}

Calcular.prototype.areaTriangulo = areaTriangulo //crear método a partir de la función

function triangulo() { //resultados para triángulo
  instanciar();
  result = numeros.areaTriangulo();
  operacion = "Area del triángulo: ";
  escribir();
}

function instanciar(){ //función que crea un objeto y asigna valores a las propiedades.
  num1 = document.calcular.num1.value;
  num2 = document.calcular.num2.value;
  numeros = new Calcular();
```

```
        numeros.n1 = num1;
        numeros.n2 = num2;
        return numeros;
      }

      function escribir() { //escribir resultados en el documento.
        texto = document.getElementById("respuesta");
        texto.innerHTML = operacion +  result;
      }
  </script>
 </head>
 <body>
  <h1>Calcular el área de círculos, rectángulos y triángulos.</h1>
  <form action="#" name="calcular">
   <p><input type="text" name="num1" size="4" /> Radio (círculos), lado1 (rectángulos), o base(triángulo)</p>
   <p><input type="text" name="num2" size="4" /> Lado2 (rectángulo), altura (triángulo)</p>
   <p>
   <input type="button" value="Rectángulo" onclick="rectangulo()" /> ...
   <input type="button" value="Círculo" onclick="circulo()" /> ...
   <input type="button" value="Triángulo" onclick="triangulo()" /> ...
   </p>
  </form>
  <h3 id="respuesta"></h3>
 </body>
</html>
```

La propiedad prototype II

La propiedad prototype en objetos predefinidos.

En la página anterior hemos visto cómo usar la propiedad prototype en objetos definidos por el programador, ahora veremos cómo usarla en los objetos predefinidos en javascript.

Podemos definir nuevos métodos y propiedades para los arrays, cadenas de texto, números, valores booleanos... Sin embargo el objeto Math no admite la propiedad prototype, ya que su función no es crear nuevos objetos, sino resolver ciertas operaciones matemáticas.

Propiedades

Crear nuevas propiedades para objetos predefinidos en javascript, se hace de la misma manera que para objetos creados por el programador, es decir, mediante la propiedad prototype, por ejemplo:

```
String.prototype.nombre = "nombre por defecto";
```

En principio se podría pensar que el código anterior añade una nueva propiedad a cualquier cadena de texto. Sin embargo **esta propiedad es de sólo lectura**

para las cadenas de texto que no estén definidas explícitamente mediante el constructor, es decir, si definimos una cadena de texto de la siguiente manera:

```
var texto = "Esta es una cadena de texto."
```

La propiedad no se puede modificar, sin embargo si la definimos mediante el objeto String de la siguiente manera:

```
var otroTexto = new String("Esta es otra cadena de texto");
```

Podremos modificar la propiedad:

```
texto.nombre = "Párrafo primero";
otroTexto.nombre = "Párrafo segundo."
```

Aquí hemos dado otro valor a la propiedad para las dos cadenas anteriores, sin embargo sólo la segunda tomará el valor nuevo, mientras que la primera seguirá teniendo el valor por defecto. Escribimos el script anterior en una página y lo comprobamos en una ventana de alerta, de la siguiente manera:

```
String.prototype.nombre = "nombre por defecto";
var texto = "Esta es una cadena de texto.";
var otroTexto = new String("Esta es otra cadena de texto");
texto.nombre = "Párrafo primero";
otroTexto.nombre = "Párrafo segundo."
ventana1 = "objeto1: "+texto+"\n propiedad: "+texto.nombre;
ventana2 = "objeto2: "+otroTexto+"\n propiedad: "+otroTexto.nombre;
alert(ventana1);
alert(ventana2);
```

El resultado serán dos ventanas de alerta en las que se muestra el objeto y su propiedad, donde se ve que en el segundo caso la propiedad se ha modificado, pero no en el primero.

Este comportamiento sucede con las clases String, Number y Boolean, sin embargo con la clase Array el comportamiento es distinto.

Para los arrays la propiedad nueva funciona siempre tanto para lectura, como para escritura, incluso en los arrays que hubiera antes de aplicar la propiedad prototype; Veámoslo con el siguiente script, que por otra parte es similar al anterior, pero con arrays:

```
var cifras = [0,1,2,3,4,5,6,7,8,9];
Array.prototype.nombre = "valor por defecto";
var estaciones = new Array("primavera","verano","otoño","invierno");
cifras.nombre = "Cifras";
estaciones.nombre = "Estaciones";
document.write("Nombre del array-> "+cifras.nombre+" : "+cifras.join("- ")+"<br>");
```

```
document.write("Nombre del array-> "+estaciones.nombre+" : "+estaciones.join("- ")+"<br>");
```

En este caso la propiedad nombre se muestra modificada en los dos arrays.

Métodos

Sin embargo tal vez lo más interesante para una programación útil puedan ser los métodos. Los métodos, funcionan siempre con todos los objetos de las clases anteriores, ya estén declarados explícitamente o no. Sin embargo la forma de trabajar es distinta de los objetos creados por nosotros, ya que los objetos predefinidos son distintos.

Cuando creamos un método para un objeto definido por el programador, usamos la referencia a las propiedades del objeto mediante this.propiedad en la función que genera el método. Sin embargo aquí lo que queremos no es acceder a las propiedades del objeto, sino al objeto en sí.

Los objetos creados por el programador son en sí objetos vacíos, el objeto en sí no contiene nada, son sus propiedades y métodos los que hacen que ese objeto tenga unos contenidos. Sin embargo los objetos predefinidos sí que tienen un contenido en sí mismos, y es a ese contenido al que se quiere acceder para modificarlo.

La forma de acceder a ese contenido al construir la función que genera el método es mediante la palabra reservada **this, sin añadirle propiedades**, y la forma más simple de trabajar con ella es igualar una variable a la palabra this:

```
cadena = this;
```

Luego ya podemos trabajar con la variable, como si fuera el propio objeto que queremos modificar.

Veamos un ejemplo. Aquí haremos una función que pone los elementos de un array entre guiones, y los pasa a mayúsculas:

```
var estaciones = ["primavera","verano","otoño","invierno"];
function guionMayus() {
    var nuevoArray = this;
    for (i in nuevoArray) {
            nuevoArray[i] = "-"+nuevoArray[i]+"-";
             nuevoArray[i] = nuevoArray[i].toUpperCase();
    }
    return nuevoArray;
 }
Array.prototype.guionMayus = guionMayus;
document.write("array original: \n"+estaciones +"<br>");
document.write("array modificado: \n"+estaciones.guionMayus()+"<br>");
```

Al igual que hacemos con los objetos definidos por el programador, primero construimos la función, y después mediante la propiedad prototype se construye el método. La diferencia es que aquí la palabra **reservada this se refiere al propio objeto** (array en este caso) que queremos modificar, el cual lo hemos asignado a una variable (nuevoArray).

Vamos a ver otro ejemplo con la clase Number. Aquí el nuevo método pone dos decimales a un número, lo redondea si tiene más y pone el símbolo del euro detrás (€):

```
function euro(){
    euros = this;
    euros = euros.toFixed(2) + "€";
    return euros;
}
Number.prototype.euro = euro;

num = 4.1732;

alert("número original: "+num);

alert("número modificado: "+num.euro());
```

Como puedes ver este es un método muy sencillo que se aplica a la claseNumber y puede ser útil para cuando los números representan dinero.