Desarrollo de Aplicaciones telemáticas
Examen final, prueba de teoría. 21 de mayo de 2024
Grado en Ingeniería en Tecnologías de la Telecomunicación
Grado en Ingeniería en Sistemas de la Telecomunicación
Universidad Rey Juan Carlos

Instrucciones:

- Rellena el fichero `~/teoria.txt` con tus datos personales y la respuesta a las siguientes preguntas.

Ejercicio 1 (4 puntos)

Indica si los siguientes selectores CSS son correctos o no. Si son correctos, describe su funcionamiento. Si no son correctos, explica brevemente por qué. Un apartado erróneo o en blanco tendrá una penalización del 50 %. Dos o más apartados erróneos o en blanco, tendrán puntuación nula.

1) `table img`

2) `table .img`

3) `p  avanzado`

4) `table.avanzado`

Respuesta

1) Correcto. Selecciona los elementos *img* dentro de elementos *table* [1].

2) Correcto. Selecciona los elementos de clase *img* descendentes de elementos *table*. Es una mala práctica usar *img* como nombre de clase, porque induce a confusión. Pero la norma lo permite.

3) Erróneo. Seleccionaría los elementos HTML de tipo *avanzado* contenidos dentro de párrafos. Pero en HTML no hay ningún elemento que se llame así. Podría ser una clase o un identificador, pero entonces el selector sería `#avanzado` o `.avanzado`

4) Correcto. Se refiere a los elementos *table* que sean de clase *avanzado*.

Ejercicio 2 (3 puntos)

Describe brevemente el *problema del gorila y el plátano* en la programación orientada a objetos tradicional. ¿Qué solución tiene esto en JavaScript?

[1] Si estuviera al revés, img table, sería un selector sin sentido, nunca seleccionaría nada porque las imágenes son de tipo void, no pueden tener contenido

Respuesta

Es un problema que se da POO basado en herencia. Se ilustra con la expresión *Yo quería solo el plátano, pero me dieron el plátano, el gorila que sujetaba el plátano y la jungla entera.* La programación orientada a objetos basada en herencia es fuertemente acoplada: los herederos de una clase reciben todas sus propiedades, junto con las propiedades de todos sus antepasados. Esto suele añadir una complejidad innecesaria[2].

La solución en JavaScript es emplear un paradigma diferente: la *programación orientada a objetos basada en prototipos*. No usa herencia, sino composición de objetos. Un objeto se crea a partir de otro, toma de este solo las propiedades necesarias. El inconveniente de este enfoque es que usarlo bien resulta complejo, y por tanto está muy poco extendido.

Ejercicio 3 (3 puntos)

¿En qué consiste la *same origin policy*?

Respuesta

Es una restricción de seguridad según la cual un programa JavaScript que se ha descargado de cierta dirección web, solo puede hacer peticiones de datos a su misma dirección web y a ninguna otra. De lo contrario, un script malicioso podría hacer peticiones de datos a otros sitios, haciéndose pasar por el usuario legítimo.

[2]El problema del código yo-yo está muy relacionado pero no es exactamente lo mismo: consiste en tener que recorrer la jerarquía de clases arriba y abajo para poder trabajar con las propiedades que necesitamos