DigitalHouse>

Certified Tech Developer
The Ultimate Degree

Front End II

Herencia, especialización y composición

Volvamos a la analogía de las plantillas para explorar este tema. Supongamos que deseamos tener fichas de películas organizadas en subgéneros. Podríamos cambiar la plantilla creada anteriormente y agregarle datos y funciones cada vez que queramos agregar otro subgénero. El problema es que estos agregados seguramente serán irrelevantes para los otros géneros y subgéneros, por lo que terminaríamos con una plantilla confusa que mezcla géneros y subgéneros y que crece constantemente con cada agregado.

Ahora, supongamos que decidimos crear una plantilla para cada subgénero. Copiaríamos la información común en cada plantilla, y agregaríamos los datos y funciones relevantes según el subgénero. El problema con esta solución es que terminaríamos duplicando información y cada vez que queramos agregar datos y funciones que son independientes del subgénero deberemos actualizar todas las plantillas o tendremos inconsistencias. Por ejemplo, supongamos que decidimos agregar el idioma de la película y tenemos 20 tipos de plantillas. Entonces, deberemos recordar agregar ese dato a las 20 plantillas, y así para cada nuevo dato o función que sea común a todas.

La programación orientada a objetos provee soluciones para estos problemas, conocidas como: **herencia, especialización y composición**. La herencia y la especialización van de la mano, mientras que la composición es la alternativa a estas.

Así, una solución es crear una clase base con datos y funciones comunes para todas las clases. Luego, crear subclases a partir de esta y que solo agreguen los nuevos tipos de datos y funciones según necesiten. Esto es lo que se conoce como herencia y especialización.

La alternativa es crear clases independientes y, cuando necesitemos datos o funcionalidad de algunas de estas clases, simplemente utilizarlas desde dentro de la clase que la requiera. Esto es composición.

Volvamos a nuestro ejemplo con plantillas y veamos cómo resolver el tema de los subgéneros.

Herencia y especialización

Supongamos que queremos crear dos subgéneros: terror y ciencia ficción. La solución consiste en crear las dos plantillas desde la plantilla base y a estas nuevas plantillas agregarles los datos nuevos. De esa manera, nuestras plantillas finales serían creadas automáticamente y lucirían así:

Película
<carga-automática-de-la-imagen-de-la-película>
Título
Año
Director
Actores
Puntaje
<click-para-abrir-en-imdb>

Terror
Género: Terror
Subgénero

a. Subplantilla de película de terror.

Película
<carga-automática-de-la-imagen-de-la-película>
Título
Año
Director
Actores
Puntaje
<click-para-abrir-en-imdb>

Ciencia-Ficción
Género: Ciencia Ficción
Subgénero

b. Subplantilla de película de ciencia-ficción.

Básicamente, eliminamos el campo género de la plantilla Película y lo pasamos a las subplantillas para que lo predefinan. Esta es una práctica muy común en la programación orientada a objetos. Además, agregamos un nuevo campo llamado subgénero que será definido en cada ficha.

Ahora, cada subplantilla puede reutilizar las dos funciones de la plantilla base, acceder a los campos de esta (Título, Año, etc.), y puede definir funciones propias que solo estarán disponibles para las fichas creadas a partir de la subplantilla. Derivar una plantilla a partir de otra es un ejemplo de herencia clásica.

Composición

Retomemos nuestro ejemplo con plantillas y supongamos que deseamos tener una lista de películas.

En lugar de crear una plantilla a partir de otra plantilla, creando una relación plantilla=>subplantilla, podríamos incluir dentro de una plantilla otras plantillas. En nuestro ejemplo, la composición nos permite crear un componente que funciona como aplicación, y este crea los componentes películas. En este caso, no hay subplantillas, no hay especificidad. Componer una plantilla a partir de otras es un ejemplo de composición.

Otro aspecto de vital importancia es que cada plantilla es responsable de presentar los campos, de darle significado a cada campo, y del acceso a cada campo.

Nuestro ejemplo de plantilla tiene todo lo que una clase tiene: estado y funciones que interactúan con esos datos. Cambiemos la palabra plantilla por clase, y ya entramos (casi sin darnos cuenta) en el mundo de la programación orientada a objetos.

¡Hasta la próxima!