Herencia en SASS

¿Cómo? ¿Herencia en SASS? ¿Es que SASS es un lenguaje basado en el paradigma POO? Bueno, no exactamente. Quizá hablar de herencia, si dejamos que se entienda por tal lo que conocemos con ese nombre en lenguajes de programación, pueda parecer un poco pretencionso. Sin embargo, en SASS sí tenemos un cierto grado de herencia que vamos a conocer en este artículo.
En realidad es, por supuesto, una "herencia" limitada a aquello para lo que existe SASS: el desarrollo de reglas CSS para hojas de estilos. Es más una forma de reutilización de código muy "sui géneris" que una auténtica herencia al uso pero, como veremos, es un concepto sumamente interesante y útil.
Y vamos a ello, que seguro que ya estás deseando entrar en materia.

LA DIRECTIVA @extend]
La directiva @extend nos permite indicarle a un selector que debe heredar las reglas CSS definidas previamente en otro selector. Esto resulta especialmente útil cuando tenemos varios selectores que comparten la mayoría de las reglas en común, habiendo sólo unas pocas diferencias.
Imagina que tienes en tu documento HTML tres botones. Cada uno de ellos tiene una funcionalidad diferente. El primero tiene la clase accessButton, el segundo tiene la clase rejectButton y el tercero tiene la clase restartButton. No vamos a entrar aquí en detalles de cual sería su operativa, o para que los tenemos. Sólo nos interesa el hecho de que los tenemos, y de que cada uno tiene una clase. La mayoría de las propiedades CSS son comunes a los tres. Lo único que varía de uno a otro en este ejemplo es el color de fondo y el texto que aparece en el botón. Por lo tanto, tiene sentido que declaremos las propiedades en una de las clases (por ejemplo, en accessButton) y las heredemos en las otras dos clases. Luego, en estas dos clases, sobrescribiremos las propiedades que son diferentes. Lo vemos en extend_1.scss:

```
@charset "UTF-8";

.accessButton {
   // Definimos las reglas CSS para una clase
   width: 10rem;
   height: 3rem;
   text-align: center;
   font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
   font-size: 1rem;
   line-height: 1rem;
   text-transform: uppercase;
   &::before {
      content: "acceso";
   }
   background-color : #32cd32;
}
```

```
.rejectButton {
    // Heremdamos todas las reglas CSS que
    // se han definido para la clase "accessButton".
    @extend .accessButton;
    // Sobrescribimos las reglas que queremos
    // cambiar para esta clase en concreto.
    &::before {
        content: "rechazo";
    }
    background-color : crimson;
}

.restartButton {
    // Heremdamos todas las reglas CSS que
    // se han definido para la clase "accessButton".
    // Como la clase "rejectButton" hereda de "accessButton"
    // Y las propiedades que queremos modificar aquí
    // las sobrescribimos después, en esta caso podríamos
    // haber heredado de la clase "rejectButton".
    @extend .accessButton;
    // Sobrescribimos las reglas que queremos
    // cambiar para esta clase en concreto.
    &::before {
        content: "reinicio";
    }
    background-color : #ffa500;
}
```

Observa las líneas resaltadas, y los comentarios del código. Cómo ves, hemos declarado todas las propiedades en el bloque de reglas CSS que corresponden a la primera clase. En las otras dos, heredamos las propiedades de la primera clase y sobrescribimos las que deseamos cambiar.

Esto funciona, pero no de la forma que nos gustaría. Observa la transpilación en extend_1.css:

```
@charset "UTF-8";

/* line 3, scss/extend_1.scss */
.accessButton, .rejectButton, .restartButton {
  width: 10rem;
  height: 3rem;
  text-align: center;
  font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
  font-size: 1rem;
  line-height: 1rem;
  text-transform: uppercase;
  background-color: #32cd32;
}

/* line 12, scss/extend_1.scss */
.accessButton::before, .rejectButton::before, .restartButton::before {
  content: "acceso";
}
```

```
/* line 18, scss/extend_1.scss */
.rejectButton {
  background-color: crimson;
}

/* line 24, scss/extend_1.scss */
.rejectButton::before {
  content: "rechazo";
}

/* line 30, scss/extend_1.scss */
.restartButton {
  background-color: #ffa500;
}

/* line 40, scss/extend_1.scss */
.restartButton::before {
  content: "reinicio";
}
```

Si examinas un poco el código CSS verás que hay cosas que, aunque funcionen bien, no son correctas desde el punto de vista de la programación. Por ejemplo, en el primer bloque ves que a las tres clases se les asigna el mismo color de fondo (mira la línea 12). Después, para dos de las clases se sobrescribe el color de fondo, como ves en las líneas de la 21 a la 23, y de la 31 a la 33.
Con el pseudo elemento ::before ocurre algo parecido. Primero se le asigna el mismo texto al contenido de las tres clases, cómo ves en las líneas de la 16 a la 18. Después se sobrescribe dicho contenido para las dos clases que heredan de la primera, cómo ves en las líneas de la 26 a la 28 y de la 36 a la 38.
Esto no queda elegante. No tiene sentido que declaremos una regla para un selector, si luego la vamos a sobrescribir directamente. Sin embargo, es una limitación que tiene la herencia en SASS. Al transpilar, no sustituye directamente las propiedades que son diferentes, sino que escribe primero las originales, y luego crea reglas nuevas para sobrescribir las que son distintas. Veamos cómo podemos arreglar este despropósito:

LOS PLACEHOLDERS]
SASS incorpora la posibilidad de crear unos bloques de código que se conocen con el nombre genérico de **placeholders**. Se declaran precediendo su nombre con el signo %. Lo bueno de los placeholders es que son bloques de código que no son directamente transpilados a CSS. Es decir. Si tu archivo SASS sólo tiene placeholders con algunas reglas CSS, al transpilar tu archivo CSS no tendrá ningún código. Son bloques de código abstractos o, como los llaman algunos, bloques "fantasma" (personalmente, no me gusta este término, pero como lo encontrarás en algunos sitios, te lo comento).
Lo que haremos, siguiendo con el ejemplo anterior, es crear un placeholder con las reglas que son comunes a las tres clases. Después, en la declaración de dichas clases usamos la directiva @extend pero, en esta ocasión, heredaremos del placeholder. Luego, a cada clase le añadimos las propiedades que son específicas de esa clase en concreto. Lo ves en extend_2.scss:

```
@charset "UTF-8";

%UserButtons{
   // Definimos las reglas CSS que serán comunes a las tres clases.
   // Las definimos en una regla placeholder.
   width: 10rem;
   height: 3rem;
   text-align: center;
   font-family: 'Franklin Gothic Medium','Arial Narrow',Arial,sans-serif;
   font-size: 1rem;
```

```
    line-height: 1rem;
    text-transform: uppercase;
}

.accessButton {
    // Heredamos del placeholder
    @extend %UserButtons;
    // Agregamos las reglas propias de esta clase.
    &::before {
        content: "acceso";
    }
    background-color : #32cd32;
}

.rejectButton {
    // Heredamos del placeholder
    @extend %UserButtons;
    // Agregamos las reglas propias de esta clase.
    &::before {
        content: "rechazo";
    }
    background-color : crimson;
}

.restartButton {
    // Heredamos del placeholder
    @extend %UserButtons;
    // Agregamos las reglas propias de esta clase.
    &::before {
        content: "reinicio";
    }
    background-color : #ffa500;
}
```

Observa las líneas resaltadas. En el primer bloque hemos declarado un placeholder llamado %UserButtons, que contiene las reglas que son comunes a las tres clases de botones. Al transpilar, el placeholder, como tal, no se reflejará de ningún modo en el CSS. Después, en las clases que queremos tener hemos heredado el placeholder (observa que el nombre también se precede aquí del signo %). Ahora las propiedades del placeholder sí son reales, y sí serán transpiladas en las clases donde se usan. Observa el resultado en extend_2.css:

```
@charset "UTF-8";

/* line 3, scss/extend_2.scss */
.accessButton, .rejectButton, .restartButton {
  width: 10rem;
  height: 3rem;
  text-align: center;
  font-family: 'Franklin Gothic Medium','Arial Narrow',Arial,sans-serif;
  font-size: 1rem;
  line-height: 1rem;
  text-transform: uppercase;
```

```
}

/* line 15, scss/extend_2.scss */
.accessButton {
  background-color: #32cd32;
}

/* line 19, scss/extend_2.scss */
.accessButton::before {
  content: "acceso";
}

/* line 25, scss/extend_2.scss */
.rejectButton {
  background-color: crimson;
}

/* line 29, scss/extend_2.scss */
.rejectButton::before {
  content: "rechazo";
}

/* line 35, scss/extend_2.scss */
.restartButton {
  background-color: #ffa500;
}

/* line 39, scss/extend_2.scss */
.restartButton::before {
  content: "reinicio";
}
```

Como ves, este código CSS es mucho más limpio y elegante, y menos confuso de interpretar que el anterior.

HERENCIA MÚLTIPLE]

Por último, debes saber que SASS permite que un set de reglas CSS herede varios orígenes, tanto si son declaraciones de reglas como si se trata de placeholders, o una combinación de ambos. Por ejemplo, una sintaxis como la siguiente sería correcta en SASS.

```
.ClaseHerenciaMultiple{
    @extend .selectorDeClase_1;
    @extend .selectorDeClase_2;
    @extend .selectorDeClase_1;
    @extend #selectorDeId_1;
    @extend %placeholder_1;
    @extend %placeholder_2;
}
```

Sin embargo, esta es una característica que rara vez necesitarás y, si llegas a usarla, debes tener presente que es posible que alguno de los orígenes de herencia podría estar sobrescribiendo propiedades declaradas en otro, lo que puede llegar a darte un resultado que no sea el que esperabas.

CONCLUYENDO]

En este artículo hemos aprendido cómo emplear la herencia en SASS, y como optimizar la transpilación final. Los códigos para que experimentes los tienes disponibles en este enlace.