Regras de Estilo @extend

O que faz a Regra de Estilo @extend?

A regra de estilo @extend nos permite gerar um código inteligente onde podemos reaproveitar os estilos de uma classe pai adicionando estes estilos ás classes filhas desse elemento pai. Esse método é o que nós conhecemos como método BEM, uma metodologia que encoraja a criação de classes, com “sub-classes” que herdam as características da classe pai e ainda implementam uma classe modificadora específica para áquele elemento.

A regra @extend veio para tornar essa metodologia fácil de ser implementada e utilizada, onde antes teríamos que criar classes e sub-classes na unha, ou utilizando extensões como o Next.js, que geram essas classes e sub-classes automaticamente, esbarrávamos em alguns problemas, como por exemplo a criação errada de sub-classes, sub-classes que não herdaram elementos das classes pai e a grande perca de tempo criando e verificando se todas as classes e sub-classes estão realmente de acordo.

Algo importante que deve ficar marcado na nossa mente é que o @extend olha para os “seletores”, ele não foca em mixins ou functions, mas em seletores e faz a sua execução sempre em foco aos seletores e seus subseletores.

Falando um pouco sobre a metodologia BEM:

Essa metodologia é aplicada na nomenclatura das classes CSS dos nossos elementos HTML.

A sigla BEM significa Block Element Modifier (Bloco, Elemento, Modificador), ela se vale de 3 pilares que formam a base dessa metodologia e também são categorias que utilizamos para dividir os nossos elementos de bloco HTML.

Vejamos como essa metodologia se aplica no exemplo abaixo:

```html
<header class="header">
 <h1 class="header__title">Bem Vindo</h1>
 <button class="header__login">Login</button>
</header>
```

Nossa primeira classe sempre será o **bloco**: `.header`

Para criarmos os **elementos**, utilizamos 2 underlines (__) após o nome do nosso bloco: `.header__title`, `.header__login`

Para criarmos os **modificadores**, utilizamos 2 traços (–) no nosso bloco ou elemento: `.header__title--highlight`, `.header__login--active`

Veja o problema que é gerado quando usamos Next.js...

O problema:

```html
<div class="error error--serious">
  Erro fatal! Fechando aplicação.
</div>
```

```css
.error {
  border: 1px #f00;
  background-color: #fdd;
}

.error--serious {
  border-width: 3px;
}
```

Perceba que temos a seleção de “error” e de “error--serious”, mas “error--serious” não possuí as formatações da classe pai.

Veja como esse problema pode ser resolvido usando @extend!!!

Abaixo temos um exemplo de uso de @extend, onde temos o seletor “error”, e dizemos ao Sass que uma determinada classe deve ser herdeira da classe com o @extend, que é a classe “--serious”. Para dizer que “--serious” é herdeira de @extend nós utilizamo o & antes dos “--” e usamos o @extend dentro do seu bloco de execução referenciando a classe pai “erro”, dessa forma: “@extend .error;”, o Sass é muito inteligente para saber que se trata de uma aplicação do método BEM...

```
.error {
  border: 1px #f00;
  background-color: #fdd;

  &--serious {
    @extend .error;
    border-width: 3px;
  }
}
```

```
.error, .error--serious {
  border: 1px #f00;
  background-color: #fdd;
}
.error--serious {
  border-width: 3px;
}
```

Perceba que do lado esquerdo é onde nós utilizamos a regra @extend, e do lado direito temos o resultado da regra @extend aplicada, perceba que no seletor de cima temos as mesmas formatações aplicadas tanto ao seletor pai quanto ao seletor filho.

A inteligência da Regra @extend!!!

Perceba o quanto a regra @extend é inteligente em comparação com o Next.js comum...

```
.content nav.sidebar {
  @extend .info;
}

// p é incompatível com nav, será ignorado
p.info {
  background-color: #dee9fc;
}
// Sass irá combinar .content dentro de .guide
.guide .info {
  border: 1px solid rgba(#000, 0.8);
  border-radius: 2px;
}
// Sass vai aplicar automaticamente main.content também
main.content .info {
  font-size: 0.8em;
}
```

```
p.info {
  background-color: #dee9fc;
}

.guide .info, .guide .content
nav.sidebar, .content .guide nav.sidebar
{
  border: 1px solid rgba(0, 0, 0, 0.8);
  border-radius: 2px;
}

main.content .info, main.content
nav.sidebar {
  font-size: 0.8em;
}
```

Perceba que do lado esquerdo nós temos a declaração da regra @extend sobre o seletor “info” para todos os seletores “info” que forem filhos de “.content” e de “nav.sidebar”, note abaixo que temos a seleção comum em Next.js para todos os seletores que seguem essa classe pai, perceba que o Next.js gera sub-classes de forma genérica. Agora do lado direito temos o resultado com o @extend, perceba que foram criados vários seletores diferentes para as mais variadas situações que possam existir dentro do código, ou seja, o @extend garante que um elemento não vá deixar de ter a sua formatação CSS aplicada.

Se não bastasse isso, ele ainda agrupa as seleções em vez de criar várias seleções diferentes para cada caso, isso torna um código muito menos inchado e mais fácil de ser carregado.