2.8. Свод правил БЭМ-а

~ 28 минут

Чтобы избавиться от сложностей при расширении функциональности, сделать код понятнее, чтобы его легко было поддерживать и делить между всеми участниками команды, в БЭМ придерживаются определённых правил. Ниже перечислим их. Будет много правил, как *не делать*, но и как *делать*, мы тоже расскажем.

Правило 1. Не использовать селектор по идентификатору

За счёт идентификатора вы указываете уникальное имя для элемента HTML. Это значит, что и стили элемента будут уникальны и переиспользовать их уже не получится.

В развивающемся проекте (то есть любом, о котором не забудут сразу, как только сделали) постоянно происходит переосмысление внешнего вида и функциональности, и никогда нельзя быть уверенными, что элемент, который сейчас встречается на странице только однажды, завтра не будет использоваться массово.

Даже такая, казалось бы, единичная вещь, как логотип или подвал.

```
/* лучше избегать подобных селекторов */
#logo {
  width: 120px;
  height: 35px;
  padding: 10px 0;
}
/* а что, если логотип на сайте будет и в шапке, и в подвале? */
```

Правило 2. Не использовать селектор по тегу

Разметка элементов страницы может поменяться со временем, к примеру, измен вложенность разделов, это повлияет на уровень заголовков. Или там, где был пр

текст, может появиться блок с подложкой. К тому же количество тегов ограничено, и если привязываться к названию тегов, то придётся переопределять множество стилей для разных элементов сайта, в которых использовались одни и те же теги.

```
<section>
  <h1>Совы могут поворачивать голову на 270° без вреда для здоровья</h1>
  ...
</section>

<!-- поменялась вложенность разделов и изменился порядок заголовков -->
<section>
  <h2>Совы могут поворачивать голову на 270° без вреда для здоровья</h2>
  ...
</section>
```

```
/* правило уже не сработает для элемента *Совы могут поворачивать...* */
h1 {
  font-size: 22px;
  line-height: 1.5;
  font-weight: 500;
}
```

Правило 3. Не использовать универсальный селектор (*)

Есть мнение, что общие стили уменьшают время разработки и сокращают размер кода. Но это не совсем верно. Во-первых, такой селектор влияет на все элементы на странице, это ограничивает перенос CSS-правил в другие проекты, ведь неизвестно как они повлияют на стили этих проектов. Во-вторых, если вы в своём проекте используете сторонние компоненты, то универсальный селектор затронет и их, а это не всегда хорошо.

```
/* плохо, устанавливает CSS-свойство всем элементам без разбора */
* {
  box-sizing: border-box;
}

/* хорошо, точечная установка CSS-свойства */
.button {
  box-sizing: border-box;
}
```

А что же делать? Какие селекторы использовать?

Только селекторы по классу, написанные в актуальной для проекта нотации и с правильным описанием блоков, элементов и модификаторов.

```
.banner__content {
  display: grid;
  align-content: center;
  justify-items: start;
}
```

О том, как правильно называть классы, мы расскажем в разделе «Названия классов по БЭМ».

Правило 4. Не использовать CSS-сброс

Про то, что такое CSS-сброс, у нас была статья в блоге: «Нормальный сброс».

Причина, по которой не стоит использовать CSS-сброс, похожа на ту, что описана в пункте про универсальный селектор. CSS-сброс — это набор глобальных CSS-правил, которые созданы для всей страницы. Эти стили влияют на всю разметку целиком, нарушают независимость компонентов и затрудняют их повторное использование.

Ресеты и нормалайзы отменяют и/или переопределяют стили, которые браузер придаёт тегам по умолчанию, а потом эти же стили придётся ещё раз переопределить для того, чтобы результат разработки соответствовал макету.

А что делать?

Определять стили блоков изолированно, так, чтобы сброс отступов по умолчанию, например, у списка определений в подвале, не повлиял бы на список определений в карточках, ведь там могут быть нужны совсем другие отступы.

Правило 5. Не использовать вложенные селекторы

Вложенные селекторы увеличивают связность кода (*к примеру, межблочный каскад*) и затрудняют повторное использование кода. А ещё такие селекторы повышают специфичность правил, и их сложнее переопределить.

```
/* межблочный каскад, блок меню и блок с логотипом */
.menu .logo {
  padding-top: 10px;
  padding-bottom: 10px;
}

/* отступы устанавливаем не для блока с логотипом, а для элемента меню */
.menu__item {
  padding-top: 10px;
  padding-bottom: 10px;
}
```

Возможные исключения

Исключением из этого правила будет стилизация для CMS (систем управления контентом), где содержимое блоков генерирует пользователь (редактор, контент-менеджер, комментатор…).

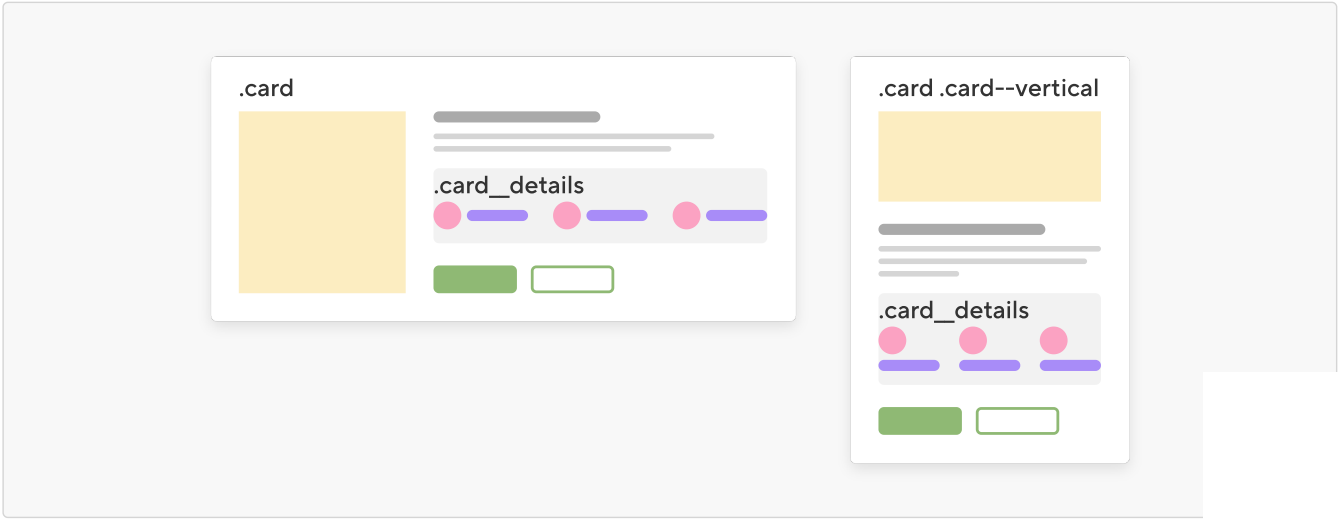
```
<div class="content">
  ... <!-- пользовательский текст -->
</div>
```

```
CSS rules:
.content a {
  ...
}

.content p {
  font-family: "Arial", sans-serif;
  text-align: center;
}
```

Мы зря учили каскад?! Никогда никакого каскада?

Почти. Использование каскада будет связано с селекторами модификаторов, а также с переопределением браузерных стилей по умолчанию и наследованием, но это — нечастое явление.

Например, у карточки есть вид горизонтальный и вертикальный, и у горизонтального данные, размеченные в список определений, выстроены в ряд, а у вертикального — в колонку. Один из вариантов будет считаться вариантом по умолчанию, второй — вариантом с модификатором, и вот этот вариант с модификатором будет переопределять вариант по умолчанию, как правило, с большей специфичностью (*в селекторе правила два класса, значит, это правило специфичнее*).



Два вида карточек с модификаторами: card (горизонтальная) и card--vertical (вертикальная)

```
<section class="card">
  <h2 class="card__title">Квартира-студия с панорамным видом 15 этаж</h2>
  ...
  <dl class="card__details">
    <dt class="card__details-item-name">гостя</dt>
    <dd class="card__details-item-value">2</dd>
    <dt class="card__details-item-name">спальня</dt>
    <dd class="card__details-item-value">1</dd>
    <dt class="card__details-item-name">кровать</dt>
    <dd class="card__details-item-value">1</dd>
    ...
  </dl>
  ...
</section>
```

```
.card__details {
  /* стили для характеристик товара */
  /* характеристики (названия и значения) выстроены в один ряд */
  display: flex;
  ...
}
```

```
<section class="card card--vertical">
  <h2 class="card__title">Квартира-студия с панорамным видом 15 этаж</h2>
  ...
  <dl class="card__details">
    <dt class="card__details-item-name">гостя</dt>
    <dd class="card__details-item-value">2</dd>
    <dt class="card__details-item-name">спальня</dt>
    <dd class="card__details-item-value">1</dd>
    <dt class="card__details-item-name">кровать</dt>
    <dd class="card__details-item-value">1</dd>
    ...
  </dl>
  ...
</section>
```

```
.card--vertical .card__details {
  /* стили для характеристик товара в карточке с вертикальной сеткой */
  /* располагаем элементы в три колонки */
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  ...
}

.card--vertical .card__details-item-name {
```

```
  /* название характеристики находится во втором ряду */
  grid-row: 2 / 3;
  ...
}

.card--vertical .card__details-item-value {
  /* значение характеристики находится в первом ряду */
  grid-row: 1 / 2;
  ...
}
```

Замечание

Иногда можно услышать, что стили определяются *за каскадом*. Это сленговое выражение, которое используется для обозначения вложенных селекторов. Например, `.content h2` читают как: «*за каскадом класса **content** определим стили для заголовков второго уровня*».

Правило 6. Не использовать комбинацию селекторов

К этому типу селекторов относятся селекторы по тегу и классу, например, `button.button`, по двум классам, `.page.current` и подобные. Такие правила становятся более специфичными, а значит, их сложнее переопределить.

Допустим, у вас есть кнопка:

```
<button class="button">Добавить в избранное</button>
```

В стилевом файле вы определяете для неё стили:

```
/* базовые стили для кнопки */
button.button {
  ...
  color: gray;
}
```

У части товаров, которые уже добавлены в избранное, эта кнопка должна быть в состоянии `--active` (*активная*). Для них мы добавляем класс и определяем правило:

```
.button--active {
  color: white;
}
```

Но оно не сработает. Его специфичность ниже правила с базовыми стилями кнопки

Ещё один пример, где хочется использовать комбинацию селекторов, селекторов по классам, — это выделение активной страницы в пагинации (*Pagination*). У активной страницы указаны два класса: `page` и `current`.

```
<!-- Постраничность -->
<ul class="pages">
  <!-- Активная страница -->
  <li class="page current">1</li>
  <li class="page">2</li>
  <li class="page">3</li>
  ...
</ul>
```

Определяем стили активной страницы:

```
/* специфичность селектора выше, чем у обычного селектора по классу, его будет сложнее переопределить */
.page.current {
  background-color: #999999;
  color: #ffffff;
}
```

Пока ничего криминального нет. Но если мы решим добавить класс для заблокированного состояния элемента с номером страницы, скажем, `disabled`, и текущая страница будет одновременно и активной, в этом случае нам будет сложнее переопределить стили и придётся искусственно утяжелять селектор в правиле.

```
<!-- Постраничность -->
<ul class="pages">
  <!-- Активная страница -->
  <li class="page current disabled">1</li>
  <li class="page">2</li>
  <li class="page">3</li>
  ...
</ul>
```

```
/* специфичности этого правила не хватает, чтобы переопределить цвет фона и цвет символов для заблокированной страницы */
.disabled {
  background-color: #eeeeee;
  color: #999999;
}
```

Если для активной страницы использовать модификатор, то шансы переопредел стили активной страницы выше:

```
<ul class="pages">
  <!-- Активная страница отмечена модификатором page__item--current -->
  <!-- Заблокированная страница отмечена модификатором page__item--disabled -->
  <li class="page page--current page--disabled">1</li>
  <li class="page">2</li>
  <li class="page">3</li>
  ...
</ul>
```

```
/* стили для активной страницы */
.page--current {
  background-color: #999999;
  color: #ffffff;
}

/* стили для заблокированной страницы */
.page--disabled {
  background-color: #eeeeee;
  color: #999999;
}
```

Возможные исключения

В редких случаях методология позволяет комбинировать селекторы тегов и классов. Едва ли не единственный такой случай — это работа с теми материалами CMS, которые формирует пользователь. Ведь пользователь вряд ли будет прописывать классы по БЭМ всем элементам своего текста. Значит, комментарий, который пользователь оставил, нужно стилизовать от селектора по тегу. Тут приходится использовать селекторы `p.lead`, `button.button`, `blockquote.quote--theme-simple` и подобные. Но это скорее вынужденная мера, для которой слишком сложно или тяжеловесно было бы придумывать другое решение.

Псевдоклассы и БЭМ

Использование псевдоклассов, таких как `:disabled`, `:hover`, `:focus`, `:checked`, `:first-child`, `:last-child`, `:nth-child` и других, в селекторах увеличивает их специфичность. Любой псевдокласс утяжеляет вес селектора ровно также, как если бы вы использовали комбинированный селектор по двум классам. Такой селектор сложнее переопределить.

> Псевдоклассы с нулевой специфичностью

Разберём несколько примеров.

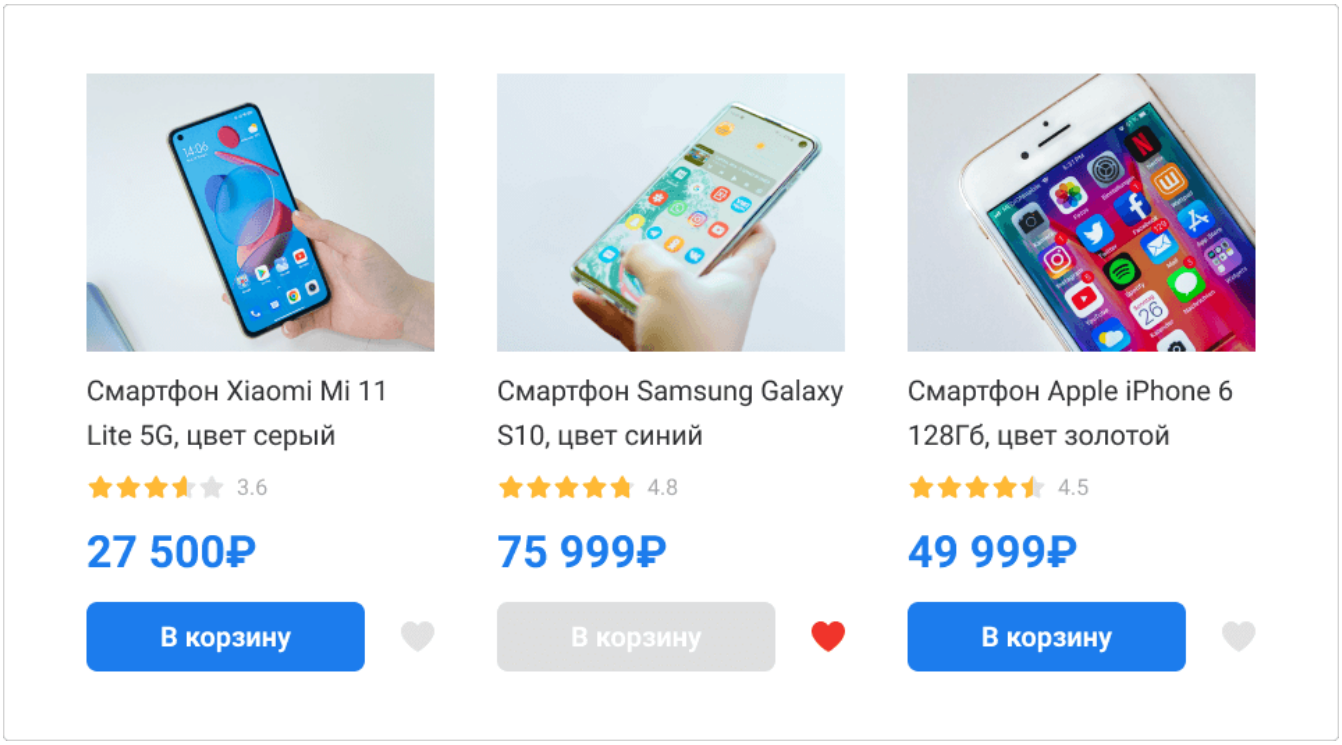
Есть псевдоклассы, которые отвечают за состояния: `:hover`, `:focus` и подобные. Без них точно не обойтись. Можно, конечно, попробовать добавлять класс через *JavaScript*. Но зачем так усложнять, если в браузер и *CSS* заложен этот функционал?

К примеру, для ссылок все используют селекторы с псевдоклассами `:hover`, `:active`, `:focus`.

```
.link {
  font-weight: 500;
  font-size: 16px;
  line-height: 1;
  color: #f85a47;
  text-decoration: none;
}

.link:hover {
  opacity: 0.7;
}

.link:active {
  opacity: 0.5;
}

.link:focus {
  opacity: 0.7;
}
```

С состоянием `:disabled` (*неактивный*) немного сложнее, оно не включается автоматически, нужен дополнительный атрибут.

Допустим, в карточке товара есть кнопка *В корзину*, её состояние (*доступна/недоступна*) определяется наличием товара в магазине/на складе: товар есть — кнопка доступна, товар отсутствует — кнопка недоступна.



Кнопка *В корзину* недоступна для товара

В этом случае доступность кнопки управляется из *JavaScript*. Но для кнопки поддерживается нативный атрибут `disabled`. Если использовать его и селектор `:disabled`, то нам как разработчикам делать ничего не нужно, весь функционал на браузере. Браузер видит, что элемент заблокирован, и не даёт ни установить фокус, ни изменить, ни нажать на него и так далее, а также при этом применятся наши стили.

```
<section class="card">
  ...
  <button class="button" type="button">В корзину</button>
</section>
...
<section class="card">
  ...
  <button class="button" type="button" disabled>В корзину</button>
</section>
```

```
.button:disabled {
  color: #b8b8cc;
  background-color: #f1f0f5;
  cursor: default;
}
```

Если бы мы опирались только на класс модификатора, пришлось бы добавлять с
писать проверки, и всё равно была бы вероятность что-то упустить. Допустим,
пользователь управляет сайтом не мышкой, а клавиатурой, а это ещё один вид пр

Можно использовать и атрибут, и модификатор, главное, синхронно ими управлять (*одновременно добавлять/удалять*):

```
<section class="card">
  ...
  <button class="button" type="button">В корзину</button>
</section>
...
<section class="card">
  ...
  <button class="button button--disabled" type="button" disabled>В
корзину</button>
</section>
```

```
.button:disabled,
.button--disabled {
  color: #b8b8cc;
  background-color: #f1f0f5;
  cursor: default;
}
```

Если атрибут `disabled` у элемента не поддерживается, а этот функционал нужен, то тут уж без класса-модификатора не обойтись.

```
<div class="rating rating--disabled" aria-label="Рейтинг отсутствует">
  <svg class="rating__star" aria-hidden="true"
xmlns="http://www.w3.org/2000/svg" viewBox="0 0 14 14">...</svg>
</div>
```

Изменить состояние `checked` (*отмечен/не отмечен*) также можно из интерфейса. Достаточно просто кликнуть на соответствующий элемент. Если у вас сложное *SPA (Single Page Application)* на каком-либо фреймворке и вы переиспользуете этот блок, да ещё и в разных модификациях, то у модификатора преимущество. А если это форма обратной связи на обычном лендинге, менять стили для отмеченного элемента проще с помощью псевдоклассов для состояний. К тому же, как и с `:hover` и `:focus`, для переключения *отмечен/не отмечен* можно обойтись вообще без *JavaScript*.

```
<form class="filters-form" action="#" method="GET">
  <label class="filters-form__option checkbox">
    <input class="checkbox__control" type="checkbox" name="brand"
value="huawei">
    <span class="checkbox__label">Huawei</span>
  </label>
  ...
</form>
```

```css
.checkbox__control {
  appearance: none;
}

.checkbox__label {
  ...
}

.checkbox__label::before {
  content: "";
  ...
}

.checkbox__control:checked + .checkbox__label::before {
  /* стили для отмеченного элемента */
}
```

Порядковые псевдоклассы `:first-child`, `:last-child`, `:nth-child` используют для стилизации отдельных элементов коллекции. Допустим, в таблице чётные строки светлые, нечётные — тёмные.

	Тариф «База»	Тариф «Стандарт»	Тариф «Про»
Стоимость	1000 ₽/год	3000 ₽/год	7000 ₽/год
Пользователи	1 пользователь	4 пользователя	10 пользователей
Объём трафика	300Гб	2Тб	20Тб
Доступ к API	–	да	да
Поддержка	круглосуточно	круглосуточно	круглосуточно

Полосатая таблица

Проще, когда модификатор родителя, то есть таблицы, определяет эту специфику. Допустим, есть несколько классов-модификаторов для самой таблицы, которые будут делать строки «полосатыми». И да, тут полосатость придётся делать теми самыми осуждаемыми порядковыми селекторами, но таблица — это особый случай.

```html
<table class="table table--striped">
  <tr class="table__row">
    <td class="table__cell">...</td>
    <td class="table__cell">...</td>
  </tr>
</table>
```

```
.table {
  background-color: #777777;
}

.table--striped .table__row:nth-child(even) {
  background-color: #eeeeee;
}
```

БЭМ исходит из максимально возможной независимости от контента, его объёма, наличия либо отсутствия и количества. А `:first-child` и `:last-child` легко ломаются при появлении других потомков — добавили семантику в таблицу (`<thead>`, `<tbody>`, `<tfoot>`) и всё, наши правила не работают.

Например, нужно отобразить насыщенным шрифтом текст в последней строке таблицы, выделить итог.

Разметка таблицы может быть такая:

```
<table class="table">
  <tr class="table__row">
    <th class="table__cell">Наименование товара</th>
    <th class="table__cell">Количество</th>
    <th class="table__cell">Цена</th>
    <th class="table__cell">Стоимость</th>
  </tr>
  <tr class="table__row">
    <td class="table__cell">Сольфеджио. Рабочая тетрадь. 5 класс. Издатель
Калинин В.В.</td>
    <td class="table__cell">1</td>
    <td class="table__cell">209.00</td>
    <td class="table__cell">209.00</td>
  </tr>
  <tr class="table__row">
    <td class="table__cell">Шоппер плюшевый, с овечкой, милый, с карманом,
мягкий, с мишками, большой, аниме, плюшевая сумка Earthman</td>
    <td class="table__cell">1</td>
    <td class="table__cell">1487.00</td>
    <td class="table__cell">1487.00</td>
  </tr>
  <tr class="table__row">
    <td class="table__cell">ИТОГ</td>
    <td class="table__cell"></td>
    <td class="table__cell"></td>
    <td class="table__cell">1696.00</td>
  </tr>
</table>
```

Используем правило с псевдоклассом `:last-child`:

```
.table__row:last-child {
  font-weight: 700;
}
```

А дальше было решено поменять разметку и добавить семантические теги для шапки, основного содержимого и подвала таблицы, к примеру, так:

```
<table class="table">
  <thead>
    <tr class="table__row">
      <th class="table__cell">Наименование товара</th>
      <th class="table__cell">Количество</th>
      <th class="table__cell">Цена</th>
      <th class="table__cell">Стоимость</th>
    </tr>
  </thead>
  <tbody>
    <tr class="table__row">
      <td class="table__cell">Сольфеджио. Рабочая тетрадь. 5 класс. Издатель
Калинин В.В.</td>
      <td class="table__cell">1</td>
      <td class="table__cell">209.00</td>
      <td class="table__cell">209.00</td>
    </tr>
    <tr class="table__row">
      <td class="table__cell">Шоппер плюшевый, с овечкой, милый, с карманом,
мягкий, с мишками, большой, аниме, плюшевая сумка Earthman</td>
      <td class="table__cell">1</td>
      <td class="table__cell">1487.00</td>
      <td class="table__cell">1487.00</td>
    </tr>
  </tbody>
  <tfoot>
    <tr class="table__row">
      <td class="table__cell">ИТОГ</td>
      <td class="table__cell"></td>
      <td class="table__cell"></td>
      <td class="table__cell">1696.00</td>
    </tr>
  </tfoot>
</table>
```

После этого правило с `:last-child` перестало работать.

Вместо псевдокласса тут лучше использовать модификатор, к примеру, `--summary`.

Ещё один кейс, когда стилизация завязана на порядковый номер, — это вертикал меню, где последний элемент без нижней границы.

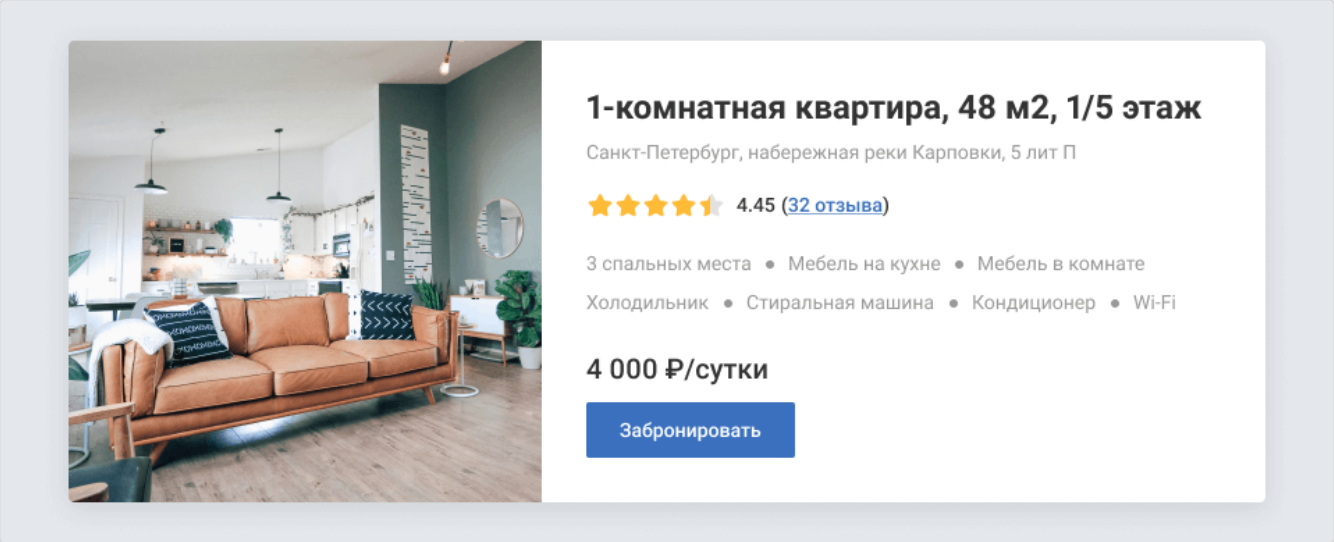
Пункты вертикального меню

Вариант с модификатором будет выглядеть так:

```
<ul class="site-navigation">
  <li class="site-navigation__item">
    <a class="site-navigation__link" href="#">Компьютеры</a>
  </li>
  <li class="site-navigation__item">
    <a class="site-navigation__link" href="#">Природа</a>
  </li>
  ...
  <li class="site-navigation__item site-navigation__item--summary">
    <a class="site-navigation__link" href="#">Абстракция</a>
  </li>
</ul>
```

И в этом случае модификатор для последнего пункта меню — это более надёжный способ стилизации. Если поменяется структура меню, селектор `.site-navigation__item:last-child` может перестать срабатывать, в отличие от модификатора. Стили уже не будут зависеть от разметки. Если блок встречается где-то ещё и с некоторыми визуальными отличиями, не придётся искусственно усложнять селектор, чтобы переопределить стили.

Есть ситуации, когда модификатор только усложнит реализацию компонента. Допустим, карточка объекта недвижимости (*или любая другая карточка товара*), где характеристики разделяются декоративными элементами (*к примеру, точками*).



Характеристики объекта недвижимости в карточке

Например, так:

```
.product-preview__capacity-param:not(:last-child)::before {
  content: "•";
}
```

Все вышеперечисленные примеры показывают, что нет чётких правил для принятия решения «псевдокласс vs модификатор». Многое зависит от типа псевдокласса, от того, что это за проект (пара-тройка страниц или большой сервис), собираетесь ли вы его поддерживать или «сделали и забыли», будет ли переиспользоваться этот элемент в проекте, будет ли элемент видоизменяться.

> Как побороть специфичность правил с псевдоклассами

Правило 7. Не использовать селектор по атрибуту

Такие правила не информативны, и неясно, к какому блоку/элементу они относятся.

К примеру, у нас есть форма поиска в шапке сайта:

```
<header>
  <form action="/" method="GET">
    <input type="text" name="search">
    <button type="submit">Найти</button>
  </form>
</header>
```

Используем селектор атрибутов для написания стилей формы:

```
header button[type=submit] {
  width: auto;
  margin-right: 20px;
}
```

По имени селектора сложно сказать, что стили относятся к форме поиска. Использование классов, к чему и призывает БЭМ, делает код более понятным. Ведь класс — это единственный селектор, который позволяет изолировать стили каждого компонента в проекте, улучшить читаемость кода и не ограничивать повторное использование самих компонентов.

Этот пример можно переписать, допустим, так:

```
<header class="page-header">
  <form class="search" action="/" method="GET">
    <input class="search__field" type="text" name="search">
    <button class="search__button button" type="submit">Найти</button>
  </form>
</header>
```

```
.search__button {
  width: auto;
  margin-right: 20px;
}
```

Иногда селекторы по атрибутам всё же добавляют в правила, к примеру, селекторы по атрибуту `href`.

Для активной ссылки в хлебных крошках определить отличный цвет текста и изменить вид курсора на обычный можно так:

```
<ol class="breadcrumbs">
  <li class="breadcrumbs__item">
    <a class="breadcrumbs__link" href="#">Главная</a>
  </li>
  <li class="breadcrumbs__item">
    <a class="breadcrumbs__link" href="#">Аренда</a>
  </li>
  <li class="breadcrumbs__item">
    <a class="breadcrumbs__link">Посуточная</a>
  </li>
</ol>
```

```
.breadcrumbs__link {
  color: #e1e3e6;
  text-decoration: none;
}

.breadcrumbs__link:not([href]) {
  color: #979899;
  cursor: default;
}
```

Или актуальные сейчас `aria-` атрибуты. В этот же пример с хлебными крошками можно добавить доступности и переписать так:

```
<ol class="breadcrumbs">
  <li class="breadcrumbs__item">
    <a class="breadcrumbs__link" href="#">Главная</a>
  </li>
  <li class="breadcrumbs__item">
    <a class="breadcrumbs__link" href="#">Аренда</a>
  </li>
  <li class="breadcrumbs__item">
    <a class="breadcrumbs__link" href="./" aria-current="page">Посуточная</a>
  </li>
</ol>
```

```
.breadcrumbs__link {
  color: #e1e3e6;
  text-decoration: none;
}

.breadcrumbs__link[aria-current] {
  color: #979899;
  cursor: default;
}
```

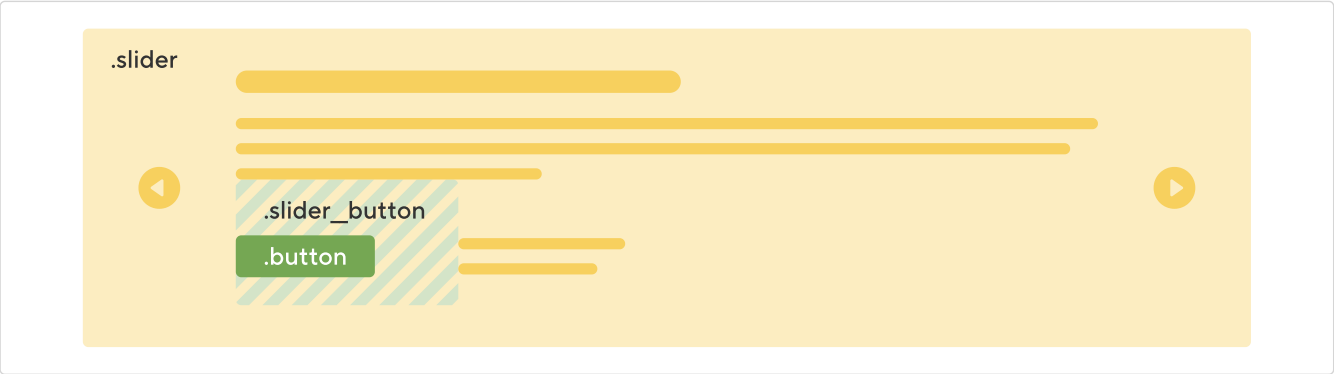
И кстати, сами авторы БЭМа дают добро на такое использование селекторов по атрибутам.

Правило 8. Не задавать внешнюю геометрию блокам

Блок, он же компонент, должен быть изолированным. Это означает, что все свои отступы он должен хранить внутри себя, а не «давить» ими на другие блоки.

Из этого выводится запрет на внешние отступы у блоков. `margin` — запрещённое свойство для блоков в БЭМ. Внешние отступы могут быть у элементов блока, если нужно. А у самого блока — только внутренние.

Например, мы назначаем слайдер блоком. У слайдера внутри есть ещё компонент-кнопка, которая в данной ситуации также является элементом слайдера (микс).
И у кнопки есть отступы снаружи, она явно отталкивается от других элементов внутри слайдера. Отступы нужно задать элементу слайдера, а не блоку-кнопке, иначе во всех других ситуациях использования у кнопки также будут внешние отступы, которые ей совсем не нужны. То же и со слайдером: вероятно, у него будут внутренние отступы, а внешние будут заданы его обёртке — элементу более крупного блока.



Блок-кнопка отталкивает соседние элементы

Замечание

Если блок состоит из элементов, важно, чтобы элементы не «вываливались» из родительского блока. К примеру, у вас есть список карточек, они выстраиваются в одну колонку, чтобы задать расстояние между карточками, и вы используете `margin-bottom`. Если у блока не предусмотрены внутренние отступы, есть шанс, что последний элемент выпадет из блока со списком карточек (речь о блочном контексте, в частности «Выпадание» внешних отступов). Можно было бы последнему элементу установить индивидуальный отступ снизу, равный 0, допустим, так: `.catalog__item:last-child { margin-bottom: 0; }`, но БЭМ не одобряет селекторы по псевдоклассам (*читайте выше в заметке «А что насчёт псевдоклассов в БЭМ»*). Подходящих решений в этом случае несколько, допустим, это может быть сетка на *CSS Grid Layout*.

Последний элемент списка «выпадает» из родительского блока

Правило 9. Не стараться определить все стили в одном CSS правиле (шрифты, сетка, декор, отступы)

Не следует на один и тот же блок навешивать стили, касающиеся всего вообще. Такие блоки ещё называют «жадными». По возможности лучше дифференцировать, это сделает код прозрачнее.

Ещё один пример с кнопками, представим, что у нас есть темы для кнопок. Их разметка:

```
<button class="button button--theme-islands">...</button>
<button class="button button--theme-islands button--active">...</button>
```

Вы решили сэкономить и описали все правила для кнопки с темой в одном правиле с селектором `.button.button--theme-islands`. Затем нужно реализовать состояние *Кнопка активна*, вы добавляете новый класс для блока `button--active`, и теперь, чтобы переопределить свойства кнопки, придётся написать селектор `.button.button--active {}`, а это не очень хорошо (искусственно усложняется селектор).

```
.button.button--theme-islands {}
.button.button--active {}
```

Лучше определить базовые стили для класса `button` и дополнить стили для тем и активной кнопки в отдельных правилах без комбинирования селекторов.

```css
/* стили для класса button */
.button {
  border: none;
  background-color: transparent;
  font-family: inherit;
}

/* Уточнение для кнопки с темой islands */
.button--theme-islands {
  background-color: #121396;
  color: #ffffff;
}

/* Уточнение для активной кнопки */
.button--active {
  background-color: #ffffff;
  color: #121396;
  border: 1px solid #121396;
}
```

Все эти правила получены в результате набивания шишек на одних и тех же проблемах. И этот список может расширяться и дополняться командами, которые выбрали БЭМ в качестве методологии вёрстки на своём проекте.

Бывают ситуации, когда какое-то из вышеперечисленных правил может быть нарушено. Но для исключения должны быть веские причины и чёткое понимание, зачем это делается. Возможно, по-другому задачу никак не решить. Можно обосновать даже наличие правила с `!important` в стилях — большой проект, куча сторонних библиотек и компонентов. Но, как вы сами понимаете, это плохая практика, и рано или поздно такое решение «выстрелит в ногу».

В некоторых компаниях и командах разработчиков список правил для CSS не ограничивается перечисленными выше правилами, он гораздо шире. К примеру, в одной компании ввели список с названиями запрещённых классов для БЭМ-сущностей. А где-то вёрстку приравнивают к полноценной разработке и к CSS применяют принципы *SOLID* из объектно-ориентированного программирования.

В частности:

— *Single responsibility* — принцип единственной ответственности для БЭМ-сущностей, чтобы не получилось, что один класс определяет кучу CSS свойств (внешний вид, позиционирование, раскладка). Блок отвечает за что-то одно, например, за сетку. У блока должна быть только одна причина для изменения.

— *Open-closed* — принцип открытости/закрытости, с помощью него запрещаетс переопределение свойств БЭМ сущностей, также этот принцип показывает, ч мы верно выбрали базовый класс, который затем при необходимости дополня

Используют принцип *DRY* (Don't Repeat Yourself, «не повторяйся»), который спасает от бездумного дублирования кода.

Не обходится и без традиционного *KISS* (Keep It Simple, Stupid, «делай проще, тупица»), ведь всегда чем проще, тем лучше и понятнее.

Получается, что свёрстанный по БЭМ сайт — это абсолютно плоская система, в которой в отдельных модулях будут расписаны стили для отдельных блоков. Эти блоки можно будет использовать в других местах на сайте вне зависимости от содержимого. А ещё вы наверняка будете знать, где лежат стили каждой конкретной детали, если их нужно будет поменять или поправить, и не нужно будет изворачиваться и что-то придумывать, чтобы переопределить стили, ведь «лишнего» каскада в системе не будет.

Прочитали главу?

Нажмите кнопку «Готово», чтобы сохранить прогресс.

Готово

У вас осталось **10** из 10 консультаций.

История




Практикум

Тренажёры

Подписка

Для команд и компаний

Учебник по PHP

Профессии

Фронтенд-разработчик

JavaScript-разработчик

Фулстек-разработчик

Курсы

HTML и CSS. Профессиональная вёрстка сайтов

HTML и CSS. Адаптивная вёрстка и автоматизация

JavaScript. Профессиональная разработка веб-интерфейсов

JavaScript. Архитектура клиентских приложений

React. Разработка сложных клиентских приложений

Node.js. Профессиональная разработка REST API

Node.js и Nest.js. Микросервисная архитектура

TypeScript. Теория типов

Алгоритмы и структуры данных

Паттерны проектирования

Webpack

Vue.js 3. Разработка клиентских приложений

Git и GitHub

Анимация для фронтендеров

Блог

С чего начать

Шпаргалки для разработчиков

Отчеты о курсах

Информация

Об Академии

О центре карьеры

Услуги

Работа наставником

Для учителей

Стать автором

Остальное

Написать нам

Мероприятия

Форум

Соглашение

Конфиденциальность

Сведения об образовательной организации

Лицензия № 4696

Участник