2.10. Названия классов по БЭМ

~ 20 минут

Название класса по БЭМ складывается из двух составляющих: *синтаксиса* — это то, как будут отделены элементы и модификаторы (двойной дефис, двойное или одиночное подчёркивание), — и *семантики*, в которой заложен смысл, предназначение блока, его миссия. Если синтаксис не важен, как говорят создатели БЭМа (*главное, чтобы все участники команды придерживались одного стиля*), то семантике уделяют особое внимание (*есть даже список запрещённых имён*).

Основная идея соглашения по именованию: имя сущности должно раскрывать её смысл и быть понятным для всех. К примеру, в CSS мы находим класс для блока `service-navigation` и понимаем, что это вспомогательная навигация. В этом подходе есть минус: названия становятся тяжеловесными. О других минусах методологии БЭМ читайте в разделе «Плюсы и минусы БЭМ».

Сейчас мы будем разбираться с тем подходом к именованию, который стал общепринятым и разошёлся по всему миру, и тем, который использует *HTML Academy*. По сути, объясним, откуда брать названия для многочисленных классов. Но сначала объясним, как называть классы не следует и почему.

Общие правила наименования для блоков, элементов и модификаторов

Имя должно быть значащим, уникальным в рамках проекта, исчерпывающим и соответствующим нотации (*синтаксису*), которой вы придерживаетесь.

Не нужно давать классам общие, не говорящие названия (такие имена ещё называют *плейсхолдерами*). Представьте, что у вас в проекте классы называются так: `titl` `content`, `section`, `inner` или `item`.

Класс `title` — это заголовок в карточке товара? Или в статье новостного блока? Из названия неясно. Можно улучшить читабельность кода, если отказаться от подобных названий классов. Кроме того, есть большая вероятность, что вам захочется назвать другой элемент таким же именем, и стили элементов будут конфликтовать. Даже если этого пока не произошло, с ростом проекта такие элементы наверняка появятся.

```
<section class="section">
  <h3 class="title">Компьютерная мышь</h3>
  <div class="content">
    <p class="text">Компьютерные технологии стали неотъемлемой частью жизни
людей. Эти технологии имеют свои корни. Возьмём, например, слово «мышка».
Компьютерная мышь совсем не то же самое, что маленький грызун, который живёт в
зданиях и полях.</p>
    <button class="action">Читать ещё!</button>
  </div>
</section>
```

Даже такое название класса, как `address`, лучше уточнить. Это адрес чего? Адрес в шапке сайта, адрес с контактами в подвале или адрес в слайдере для блока с рекламой?

Не лучшим выбором будут названия классов, в которых определяется внешний вид сущности: `uppercase`, `text-center`, `button-red`, `button-top`, `button-bottom`, `button-left` и `button-right`. Также плохой практикой считается завязывать наименования на расположение сущности на странице. Её внешний вид может измениться (у компании изменился фирменный стиль и заказали новый брендбук, пришлось менять дизайн страниц), как и расположение (в десктопной версии элементы чаще всего располагаются иначе, чем в мобильной).

К примеру, в первоначальном дизайне страницы кнопка первичного действия была одного цвета. После обновления дизайна эта же кнопка стала уже другого цвета, плюс изменилась в целом раскладка блока с этой кнопкой:

Изменили дизайн страницы, кнопки действия поменяли цвета

> У меня был случай доработки чужого проекта, где вначале акцентный цвет для кнопок был синим и название класса `button--blue` смотрелось вполне себе логично. Но потом решили поменять цвет на зеленый. А для второго акцента с красного на желтый. Вы бы только знали, как рвался мой мозг, когда я переписывала цвета в стилях... `blue` — зеленый, `red` — желтый... А переписать в огромном проекте всё это в разметке, чтоб не рвать мозг, просто не позволяло время. Проблемы бы не было, будь не `button--blue`, а `button--accent`, не `button--red`, а `button--secondary`...

Также не стоит в названии класса привязываться только к контенту. Допустим, у вас на странице есть кнопки-ссылки *Перейти в избранное*, *Загрузить ещё* и *Читать* для перехода на полную версию статьи или новости. В интерфейсе эти элементы выглядят одинаково, как обычные и акцентные кнопки. В этом случае название класса для обычной кнопки можно установить как `button` или `button--secondary`, а для акцентной — `button--primary` или `button--accent` вместо `button--favorite`, `button--more`. Слишком завязанные на контент наименования могут быть и такими: `track`, `movie`, `film`, `song` (*проект про музыку и кино*), `icecream` (*магазин мороженого*). Лучше использовать имя `product-card`. Так ваш проект для интернет-магазина можно будет использовать для продажи не только мороженого, музыки и фильмов, но и чего угодно, начиная с цветов, заканчивая мебелью. Старайтесь подбирать более универсальные названия.

Имя сущности должно соответствовать её сути. К примеру, в классе модификатора `card--theme-dark` не должно быть изменений внутренних отступов, установки позиционирования и подобного. Например, так:

```
.card--theme-dark {
  padding: 20px 30px; /* вы бы предположили по названию класса, что они здесь окажутся? */
  color: #ffffff;
  background-color: #3f3f3c;
  width: 350px;
}
```

Оставляем стили только для темы, а именно цвет шрифта и цвет фона:

```
.card--theme-dark {
  color: #ffffff;
  background-color: #3f3f3c;
}
```

Названия классов для блоков

Когда к нам приходит макет, нужно обязательно провести предварительную аналитическую работу, чтобы потом смочь выделить блоки и придумать им имена. По наименованию должно быть понятно, как его использовать.

Идеальная версия имени для блока относится к бизнес-словарю (`product`, `service`, `payment`, `warning`, `history`, `statistics`, `cover`, `event`) или к разработке интерфейсов (`header`, `footer`, `card`, `button`, `page`, `logo`). В случае если нужно разграничивать разные виды компонентов разработки, имя блока может относиться и к тому и к другому (`subscribing-form`, `site-navigation`, `lang-switcher`).

Ещё примеры удачных имён:

— Бизнес-термины: `product`, `service`, `history`, `statistics`, `search`, `socials`, `copyright`, `payment`, `meta`, `features`, `benefits`, `cart`, `feedback`, `advertisement`, `catalog`, `banner`.

— Термины из разработки интерфейсов: `header`, `footer`, `card`, `button`, `page`, `aside`, `logo`, `warning`, `error`, `form`, `content` (*как текстовый элемент*), `cover`, `event`, `avatar`, `slider`, `breadcrumbs`, `pagination`, `popup`, `modal`, `tooltip`, `preview`, `overlay`, `dropdown`.

Какие имена лучше не использовать вовсе для блоков? Плейсхолдеры, которые ничего не означают сами по себе (`item`, `case`, `object`). Русские, написанные латиницей (`shtuka`), потому что нет гарантий, что все, кто будет поддерживать этот проект кроме и после вас, будут русскоязычными.

Замечание

`container` — имя-плейсхолдер, с ходу и не скажешь, какие правила в нём лежат. Но именно такое имя очень часто встречается для обозначения контейнера области с контентом, и это не только в наших учебных материалах, но и в реальных проектах.

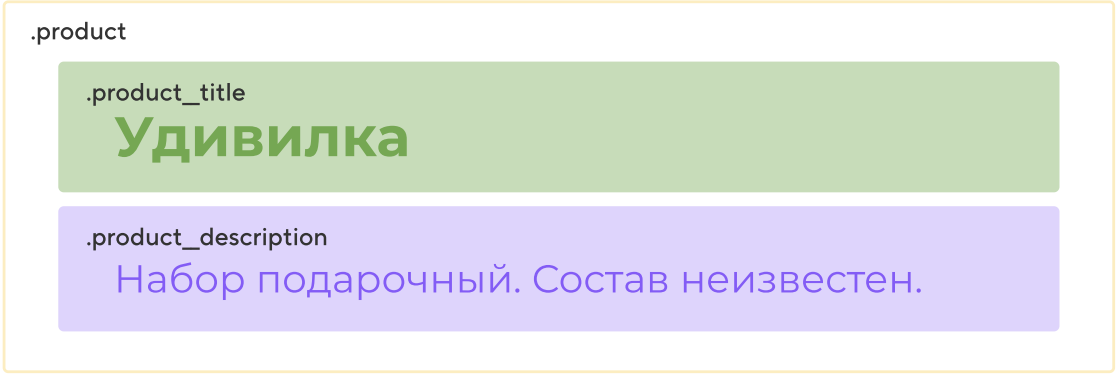
Названия классов для элементов

Элементы блока в названии класса содержат название их родительского блока. Название блока — это что-то вроде *namespace* (пространство имён) для элементов.

Пространство имён — это некое абстрактное хранилище или окружение, которое существует для логической группировки уникальных идентификаторов (то есть имён классов).

Например, у нас есть карточка товара, зададим ей пространство имён `product`. Чтобы показать, что заголовок товара является частью этой карточки, мы ему задаём на `product__title`, отделяя имя элемента от имени блока двойным подчёркивание вы сразу будете видеть в коде и стилях, как организована архитектура блока.

```
<section class="product">
  <h2 class="product__title">Удивилка</h2>
  <p class="product__description">Набор подарочный. Состав неизвестен.</p>
</section>
```



Пространство имён для элементов

Читается это так: элемент `title` блока `product` (*product__title*), элемент `description` блока `product` (*product__description*).

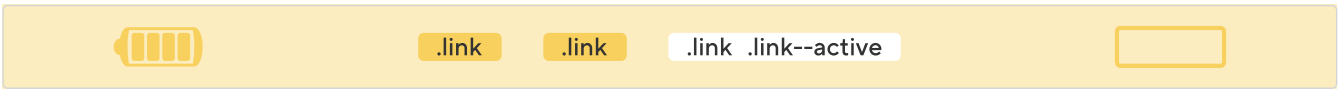
Ещё примеры: `product__button`, `history__destination`, `history__label`, `history__description` и подобные.

Обычно это слова из разработки или редакторской области, ведь они обозначают не самостоятельный блок, а то, что не функционально само по себе.

Названия классов для модификаторов

Модификация — это изменение изначального состояния, как бы очевидно это ни звучало. Значит, сначала нужно выяснить, какое состояние — исходное (обычно мы его называем *базовым*).

Например, для элементов главного меню это довольно очевидно: все ссылки обычные, а одна, которая обозначает страницу, которую мы сейчас просматриваем, другая. Следует назвать её активной или текущей (`--active`, `--current`), а не красной или подчёркнутой.



Пункт меню для активной или текущей страницы

Это и есть связь имени с задачей, функцией. Связывать имена с положением или внешним видом небезопасно: сайты изменчивы, и положение, и цвет, и размер,

и насыщенность, и **любая другая декоративная или относительная характеристика могут измениться**.

Очень часто сначала приходится погрузиться в дизайн-систему будущего сайта, провести анализ, почему, например, некоторые кнопки красные, а некоторые — большие. И во время анализа оказывается, что у кнопок разные типы действий, одна — про покупку или лидогенерацию (кнопки «первичного действия»), другая — про переход. В любом случае лучше не привязывать название модификатора к окружению. Если вы выяснили, что кнопки со скруглёнными краями и синей обводкой в макете встречаются только в каталогах разных видов, всё равно лучше не делать модификатор `--catalog`. Заказчику в любой миг может прийти в голову использовать такую кнопку в рекламном блоке, и тогда поиск её стилей станет сложнее. Что касается кнопок, то одни из самых безопасных модификаторов, которые мы видели, были либо сугубо функциональными, как `--checked`, `--danger`, либо отсылающими к уровню применения, как `--primary`, `--secondary`.

Если говорить иначе, то блок — это пространство имён, а модификатор задаёт особенности реализации.

Хорошие БЭМ-модификаторы: `level`, `theme`, `view`, `size` (вроде `size-xl`, `size-xs`), `direction`, `type-link`, `checked`, `disabled`.

Спорным именем для модификатора считается `hidden` (скрытый блок/элемент). Зачем блоку хотеть быть спрятанным? Но есть, к примеру, модальные окна, всплывающие меню, которые нужно скрывать. В этом случае можно рассуждать так:

Что делает наш модификатор? Показывает различные вариации и состояния блока. Скрытность — естественное состояние этого блока, базовое, состояние по умолчанию. Он таким родился. А видимость — это его особенное состояние, активное.

Поэтому заводим класс, который отвечает за «активность» (по факту — видимость).

```
<!-- модальное окно в состоянии по умолчанию -->
<section class="modal">
...
</section>
<!-- модальное окно в состоянии активно, то есть показывается на экране -->
<section class="modal modal--active">
...
</section>
```

```
.modal {
  ...
  display: none;
}
```

```
.modal--active {
  display: block;
}
```

Управлять этим удобно в любом масштабе, при любом количестве окон.

Ещё один неоднозначный пример модификаторов. Вы наверняка видели всплывающие подсказки с кратким описанием, зачем нужна та или иная кнопка или что означает незнакомое слово, которое используется в тексте. Подсказка может выводиться над или под элементом, слева или справа от него. Вполне логично назвать модификатор, который отвечает за положение блока, `--top`, `--bottom`, `--left`, `--right`.

```
<!-- всплывающая подсказка показывается сверху -->
<div class="tooltip tooltip--top">
...
</div>

<!-- всплывающая подсказка показывается снизу -->
<div class="tooltip tooltip--bottom">
...
</div>
```

А как бы вы назвали модификаторы для подобных всплывающих подсказок?

Часто для обозначения модификаторов-состояний используют названия, которые начинаются с префиксов `is-` и `has-` (и, возможно, с префикса `with-`). Это пространство имён перекочевало из SMACSS, где оно применяется для отслеживания состояний элементов пользовательского интерфейса.

Такие модификаторы читаются лучше по сравнению с обычными.

Оцените сами, какое название модификатора понятнее: `modal--opened` или `modal--is-open`, `button--icon` или `button--has-icon`.

Запрещённые БЭМ-модификаторы

Не лучшими именами для модификаторов будут:

— модификатор `for` или принадлежность чему-либо `button--for-dialog`;

— модификатор `on` или принадлежность чему-либо `nav--on-index` (*хотя этот префикс может встречаться в проектах*).

Модификатор следует всегда писать как характеристику блока или элемента. У в должно получиться что-то вроде `submit-form submit-form--ready`, или `site-navigation__menu-link site-navigation__menu-link--active`.

Ни в коем случае не должно появиться никаких «оторванных» модификаторов вроде `active` для активного блока/элемента, ведь тогда он будет ещё и как бы отдельным блоком. Допустим, так бы мог выглядеть *антипример* с выделением активной страницы в пагинации:

```
<ul class="page-section__pagination pagination">
  <li class="pagination__item">
    <a class="pagination__link active" href="#">1</a>
  </li>
  <li class="pagination__item">
    <a class="pagination__link" href="#">2</a>
  </li>
  <li class="pagination__item">
    <a class="pagination__link" href="#">3</a>
  </li>
  ...
</ul>
```

Но и тут может быть исключение: когда один и тот же стиль, который отвечает за изменение внешнего вида, как класс (или блок) примешивается (миксуется) к разным HTML-элементам (допустим, все выделенные блоки на странице имеют один класс `highlight`).

```
<body class="page">
  <header class="page__header page-header">
    <nav class="page__nav">
      ...
      <ul class="site-navigation">
        <li class="site-navigation__item">
          <a class="site-navigation__link" href="#">О нас</a>
        </li>
        <li class="site-navigation__item">
          <a class="site-navigation__link highlight" href="#">Sale</a>
        </li>
        ...
      </ul>
    </nav>
  </header>
  <main class="page__content">
    ...
    <section class="promo">
      ...
      <p class="promo__text highlight">Скидки до 70%</p>
      ...
    </section>
  </main>
...
```

Хорошо, когда из имени модификатора понятно, что именно он изменяет, модифі

```
<header class="page-header">
  <nav class="main-nav main-nav--inner">
  <!-- --inner — не очень понятный модификатор. «Внутреннее» что? -->
  </nav>
</header>
```

Лучше так:

```
<header class="page-header">
  <!-- Модификация главной навигации для внутренней страницы -->
  <nav class="main-nav main-nav--inner-page">
  </nav>
</header>
```

В CSS-правилах для модификаторов мы не можем использовать внешние отступы и вообще любую внешнюю геометрию. Это допустимо делать только через *миксование* и для элементов!

Ещё одно общее правило для модификатора: модификатор БЭМ-блока должен быть направлен внутрь блока, а не наружу. То есть он влияет на вид этого блока, а не на другие блоки или взаимодействие с ними.

> А что с наименованием в классическом *БЭМ* (для справки)

Стиль именования классов Академии

Мы в *Академии* используем нотацию, близкую к общепринятой, — стиль *Two Dashes* (или стиль *Гарри Робертса*). Многим нравится формат модификаторов через `--`, считается, что он читабельней. За рубежом используют его.

Синтаксис:

`block-name__elem-name--mod-name--mod-val: button, button--theme-primary`

`block__element--modifier: menu__item--active, button__icon--size-xl`

```
<ul class="breadcrumbs">
  <li class="breadcrumbs__item">
    <a class="breadcrumbs__link" href="#">Главная</a>
  </li>
  <li class="breadcrumbs__item">
    <a class="breadcrumbs__link" href="#">Курсы</a>
  </li>
```

```
  <li class="breadcrumbs__item">
    <a class="breadcrumbs__link breadcrumbs__link--active" href="#">Python</a>
  </li>
</ul>
```

Префиксы **b-** и другие для блоков у нас не прижились.

Использование булевых модификаторов и модификаторов типа ключ-значение

В классическом БЭМ выделяются два типа модификаторов — булевые, которые используют, когда важно только наличие или отсутствие модификатора, а его значение несущественно, и модификаторы *ключ-значение*, когда значение модификатора важно. И в классическом БЭМ с его нотацией имя модификатора и его значение начинается с `_` (подчёркивания):

— `имя-блока__имя-элемента_имя-модификатора` для булевых,

— `имя-блока__имя-элемента_имя-модификатора_значение-модификатора` для модификаторов *ключ-значение*.

Историческая справка

Некоторое время назад в сообществе Академии возник спор, какой разделитель предпочтительнее для модификатора типа *ключ-значение*.

В нашей системе модификатор начинается с `--` (double dash) независимо от того, какого он типа, а значение модификатора, равно как и составное имя, пишется через символ `-` (dash).

Было мнение, что если ключ модификатора или его значение состоит из нескольких слов, то для модификатров типа *ключ-значение* может быть сложно определить, что есть ключ, а что — значение.

> `block__elem--m-o-d-i-f-i-e-r` — что из этого название, а что значение модификатора?

Вариант, который предлагается в документации по БЭМ, где разделитель ключа и значения — `--` (double dash), также никого не устроил.

> `block-name__elem-name--mod-name--mod-value` — это какой-то шампур с шашлыками.

Поэтому пришли к соглашению, что если уж кто-то хочет *ключ-значение*, то можно использовать одинарное подчёркивание для отделения значения. То есть использовать вариант `block-name__elem-name--mod-name_mod-value`, в котором более явное отделение значения от названия.

Плюс этого варианта ещё и в том, что сразу видно: в проекте используют модификаторы типа *ключ-значение* (придерживаются концепции `key/value`).

Всё же в своих проектах для модификаторов *ключ-значение* мы используем символ `-` (dash).

Например, булевые модификаторы `modal--active`, `benefit__img--cropping-top-right`, модификаторы *ключ-значение* `button--size-xl`, `logo--theme-white`.

Но мы не против варианта с символом `_` (одиночное подчёркивание), если это улучшает читаемость кода.

Прочитали главу?

Нажмите кнопку «Готово», чтобы сохранить прогресс.

Готово

Если вы обнаружили ошибку или неработающую ссылку, выделите ее и нажмите Ctrl + Enter

Поиск по материалам

Git

Все материалы

В самом начале

- [ ] Пройдите опрос
- [ ] Укажите персональные данные
- [ ] Изучите регламент
- [ ] Прочитайте FAQ
- [ ] Добавьте свой Гитхаб
- [ ] Выберите наставника
- [ ] Создайте проект

Мой наставник

Выбрать наставника

Работа с наставником

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

Услуги

Работа наставником

Для учителей

Стать автором

Информация

Об Академии

О центре карьеры

Остальное

Написать нам

Мероприятия

Форум

Участник