**Теория к практическому занятию №11**

**«Методология БЭМ»**

БЭМ (Блок, Элемент, Модификатор) — компонентный подход к веб-разработке. В его основе лежит принцип разделения интерфейса на независимые блоки. Он позволяет легко и быстро разрабатывать интерфейсы любой сложности и повторно использовать существующий код, избегая «Copy-Paste».

**Содержание**

* Блок
* Элемент
* Когда создавать блок, когда — элемент?
* Модификатор
* Микс
* Файловая структура

**Блок**

Функционально независимый компонент страницы, который может быть повторно использован. В HTML блоки представлены атрибутом class.

Особенности:

* Название блока характеризует смысл («что это?» — «меню»: menu, «кнопка»: button), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).

**Пример**

<!-- Верно. Семантически осмысленный блок `error` -->

<div class="error"></div>

<!-- Неверно. Описывается внешний вид -->

<div class="red-text"></div>

* Блок не должен влиять на свое окружение, т. е. блоку не следует задавать внешнюю геометрию (в виде отступов, границ, влияющих на размеры) и позиционирование.
* В CSS по БЭМ также не рекомендуется использовать селекторы по тегам или id.

Таким образом обеспечивается независимость, при которой возможно повторное использование или перенос блоков с места на место.

**Принцип работы с блоками**

**Вложенность**

* Блоки можно вкладывать друг в друга.
* Допустима любая вложенность блоков.

**Пример**

<!-- Блок `header` -->

<header class="header">

<!-- Вложенный блок `logo` -->

<div class="logo"></div>

<!-- Вложенный блок `search-form` -->

<form class="search-form"></form>

</header>

**Элемент**

Составная часть блока, которая не может использоваться в отрыве от него.

Особенности:

* Название элемента характеризует смысл («что это?» — «пункт»: item, «текст»: text), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).
* Структура полного имени элемента соответствует схеме: имя-блока\_\_имя-элемента. Имя элемента отделяется от имени блока двумя подчеркиваниями (\_\_).

**Пример**

<!-- Блок `search-form` -->

<form class="search-form">

<!-- Элемент `input` блока `search-form` -->

<input class="search-form\_\_input">

<!-- Элемент `button` блока `search-form` -->

<button class="search-form\_\_button">Найти</button>

</form>

**Принципы работы с элементами**

* Вложенность
* Принадлежность
* Необязательность

**Вложенность**

* Элементы можно вкладывать друг в друга.
* Допустима любая вложенность элементов.
* Элемент — всегда часть блока, а не другого элемента. Это означает, что в названии элементов нельзя прописывать иерархию вида block\_\_elem1\_\_elem2.

**Пример**

<!--

Верно. Структура полного имени элементов соответствует схеме:

`имя-блока\_\_имя-элемента`

-->

<form class="search-form">

<div class="search-form\_\_content">

<input class="search-form\_\_input">

<button class="search-form\_\_button">Найти</button>

</div>

</form>

<!--

Неверно. Структура полного имени элементов не соответствует схеме:

`имя-блока\_\_имя-элемента`

-->

<form class="search-form">

<div class="search-form\_\_content">

<!--

Рекомендуется:

`search-form\_\_input` или `search-form\_\_content-input`

-->

<input class="search-form\_\_content\_\_input">

<!--

Рекомендуется:

`search-form\_\_button` или `search-form\_\_content-button`

-->

<button class="search-form\_\_content\_\_button">Найти</button>

</div>

</form>

Имя блока задает пространство имен, которое гарантирует зависимость элементов от блока (block\_\_elem).

Блок может иметь вложенную структуру элементов в DOM-дереве:

**Пример**

<div class="block">

<div class="block\_\_elem1">

<div class="block\_\_elem2">

<div class="block\_\_elem3"></div>

</div>

</div>

</div>

Однако эта же структура блока в методологии БЭМ всегда будет представлена плоским списком элементов:

**Пример**

.block {}

.block\_\_elem1 {}

.block\_\_elem2 {}

.block\_\_elem3 {}

Это позволяет изменять DOM-структуру блока без внесения правок в коде каждого отдельного элемента:

**Пример**

<div class="block">

<div class="block\_\_elem1">

<div class="block\_\_elem2"></div>

</div>

<div class="block\_\_elem3"></div>

</div>

Структура блока меняется, а правила для элементов и их названия остаются прежними.

**Принадлежность**

Элемент — **всегда часть блока** и не должен использоваться отдельно от него.

**Пример**

<!-- Верно. Элементы лежат внутри блока `search-form` -->

<!-- Блок `search-form` -->

<form class="search-form">

<!-- Элемент `input` блока `search-form` -->

<input class="search-form\_\_input">

<!-- Элемент `button` блока `search-form` -->

<button class="search-form\_\_button">Найти</button>

</form>

<!-- Неверно. Элементы лежат вне контекста блока `search-form` -->

<!-- Блок `search-form` -->

<form class="search-form">

</form>

<!-- Элемент `input` блока `search-form` -->

<input class="search-form\_\_input">

<!-- Элемент `button` блока `search-form` -->

<button class="search-form\_\_button">Найти</button>

**Необязательность**

Элемент — необязательный компонент блока. Не у всех блоков должны быть элементы.

**Пример**

<!-- Блок `search-form` -->

<div class="search-form">

<!-- Блок `input` -->

<input class="input">

<!-- Блок `button` -->

<button class="button">Найти</button>

</div>

**Когда создавать блок, когда — элемент?**

**Создавайте блок**

Если фрагмент кода может использоваться повторно и не зависит от реализации других компонентов страницы.

**Создавайте элемент**

Если фрагмент кода не может использоваться самостоятельно, без родительской сущности (блока).

Исключение составляют элементы, реализация которых для упрощения разработки требует разделения на более мелкие части — подэлементы. В БЭМ-методологии нельзя создавать элементы элементов. В подобном случае вместо элемента необходимо создавать служебный блок.

**Модификатор**

Cущность, определяющая внешний вид, состояние или поведение блока либо элемента.

Особенности:

* Название модификатора характеризует внешний вид («какой размер?», «какая тема?» и т. п. — «размер»: size\_s, «тема»: theme\_islands), состояние («чем отличается от прочих?» — «отключен»: disabled, «фокусированный»: focused) и поведение («как ведет себя?», «как взаимодействует с пользователем?» — «направление»: directions\_left-top).
* Имя модификатора отделяется от имени блока или элемента одним подчеркиванием (\_).

**Типы модификаторов**

**Булевый**

* Используют, когда важно только наличие или отсутствие модификатора, а его значение несущественно. Например, «отключен»: disabled. Считается, что при наличии булевого модификатора у сущности его значение равно true.
* Структура полного имени модификатора соответствует схеме:
  + имя-блока\_имя-модификатора;
  + имя-блока\_\_имя-элемента\_имя-модификатора.

**Пример**

<!-- Блок `search-form` имеет булевый модификатор `focused` -->

<form class="search-form search-form\_focused">

<input class="search-form\_\_input">

<!-- Элемент `button` имеет булевый модификатор `disabled` -->

<button class="search-form\_\_button search-form\_\_button\_disabled">Найти</button>

</form>

**Ключ-значение**

* Используют, когда важно значение модификатора. Например, «меню с темой оформления islands»: menu\_theme\_islands.
* Структура полного имени модификатора соответствует схеме:
  + имя-блока\_имя-модификатора\_значение-модификатора;
  + имя-блока\_\_имя-элемента\_имя-модификатора\_значение-модификатора.

**Пример**

<!-- Блок `search-form` имеет модификатор `theme` со значением `islands` -->

<form class="search-form search-form\_theme\_islands">

<input class="search-form\_\_input">

<!-- Элемент `button` имеет модификатор `size` со значением `m` -->

<button class="search-form\_\_button search-form\_\_button\_size\_m">Найти</button>

</form>

<!--

Невозможно одновременно использовать два одинаковых модификатора

с разными значениями

-->

<form class="search-form

search-form\_theme\_islands

search-form\_theme\_lite">

<input class="search-form\_\_input">

<button class="search-form\_\_button

search-form\_\_button\_size\_s

search-form\_\_button\_size\_m">

Найти

</button>

</form>

**Принципы работы с модификаторами**

**Модификатор нельзя использовать самостоятельно**

С точки зрения БЭМ-методологии модификатор не может использоваться в отрыве от модифицируемого блока или элемента. Модификатор должен изменять вид, поведение или состояние сущности, а не заменять ее.

**Пример**

<!-- Верно. Блок `search-form` имеет модификатор `theme` со значением `islands`-->

<form class="search-form search-form\_theme\_islands">

<input class="search-form\_\_input">

<button class="search-form\_\_button">Найти</button>

</form>

<!-- Неверно. Отсутствует модифицируемый класс `search-form` -->

<form class="search-form\_theme\_islands">

<input class="search-form\_\_input">

<button class="search-form\_\_button">Найти</button>

</form>

**Микс**

Прием, позволяющий использовать разные БЭМ-сущности на одном DOM-узле.

Миксы позволяют:

* совмещать поведение и стили нескольких сущностей без дублирования кода;
* создавать семантически новые компоненты интерфейса на основе имеющихся.

**Пример**

<!-- Блок `header` -->

<div class="header">

<!-- К блоку `search-form` примиксован элемент `search-form` блока `header`-->

<div class="search-form header\_\_search-form"></div>

</div>

В данном примере мы совместили поведение и стили блока search-form и элемента search-form блока header. Такой подход позволяет нам задать внешнюю геометрию и позиционирование в элементе header\_\_search-form, а сам блок search-form оставить универсальным. Таким образом, блок можно использовать в любом другом окружении, потому что он не специфицирует никакие отступы. Это позволяет нам говорить о его независимости.

**Файловая структура**

Принятый в методологии БЭМ компонентный подход применяется и к организации проектов в файловой структуре. Реализации блоков, элементов и модификаторов делятся на независимые файлы-технологии, что позволяет нам подключать их опционально.

Особенности:

* Один блок — одна директория.
* Имена блока и его директории совпадают. Например, блок header — директория header/, блок menu — директория menu/.
* Реализация блока разделяется на отдельные файлы-технологии. Например, header.css, header.js.
* Директория блока является корневой для поддиректорий соответствующих ему элементов и модификаторов.
* Имена директорий элементов начинаются с двойного подчеркивания (\_\_). Например, header/\_\_logo/, menu/\_\_item/.
* Имена директорий модификаторов начинаются с одинарного подчеркивания (\_). Например, header/\_fixed/, menu/\_theme\_islands/.
* Реализации элементов и модификаторов разделяются на отдельные файлы-технологии. Например, header\_\_input.js, header\_theme\_islands.css.

**Пример**

search-form/ # Директория блока search-form

\_\_input/ # Поддиректория элемента search-form\_\_input

search-form\_\_input.css # Реализация элемента search-form\_\_input

# в технологии CSS

search-form\_\_input.js # Реализация элемента search-form\_\_input

# в технологии JavaScript

\_\_button/ # Поддиректория элемента search-form\_\_button

search-form\_\_button.css

search-form\_\_button.js

\_theme/ # Поддиректория модификатора

# search-form\_theme

search-form\_theme\_islands.css # Реализация блока search-form, имеющего

# модификатор theme со значением islands

# в технологии CSS

search-form\_theme\_lite.css # Реализация блока search-form, имеющего

# модификатор theme со значением lite

# в технологии CSS

search-form.css # Реализация блока search-form

# в технологии CSS

search-form.js # Реализация блока search-form

# в технологии JavaScript

Такая файловая структура позволяет легко поддерживать и повторно использовать код.

Разветвленная файловая структура предполагает, что в production код будет собираться в общие файлы проекта.