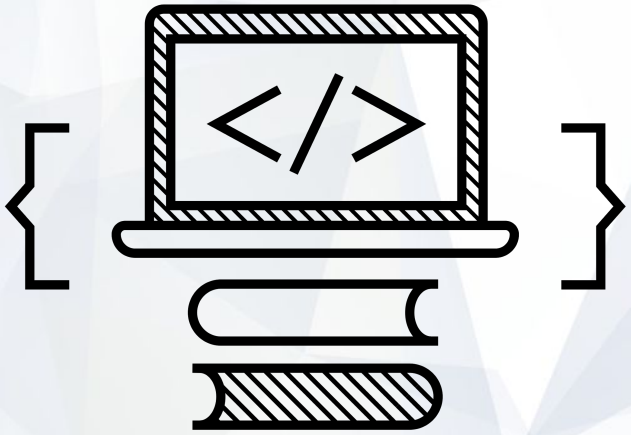
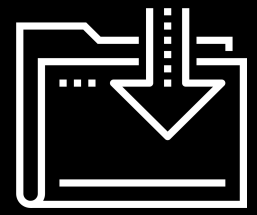


ECMAScript Modules

Course: Java

S1



Learning Outcomes

By the end of this lesson, you will be able to:

01 Export a definition.

02 Import a definition.

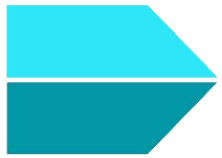
03 Export a default definition or expression.

04 Import a default definition or expression.

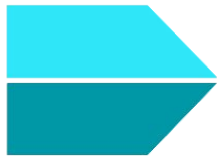
05 Use aliases for both exports and imports.

ECMAScript Modules

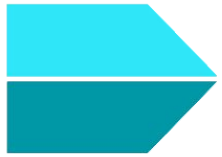
ECMAScript Modules



ES modules are the future of JavaScript modules.



They are part of the ES specification.



The latest browsers and Node.js support them.

- This makes the ES module specification the first module specification to be used both in the browser and on the server.

Enabling ES Modules

- Node.js uses CommonJS by default. We enable ES modules by setting `type` to `module` in `package.json`.
- React uses ES modules.

```
{
    "name": "tic-tac-toe",
    "type": "module", // HERE
    "version": "1.0.0",
    "description": "Tic Tac Toe Game",
    "main": "index.js"
    // <snip>
  }
```

{

The export Keyword

- The `export` keyword makes a definition available externally. Definitions can be functions, objects, classes, or values.
- The keyword is valid in four locations:
    1. Before a definition
    2. Before a default
    3. Before an export group
    4. Before a `from` "module" for re-exporting

```
export const FIVE = 5;

// private function
function add(a, b) {
    return a + b;
}

export function addFive(a) {
    return add(a, FIVE);
}
```

import * as [name]

The `import` keyword:

- Imports a definition that's been exported from another module.
- Has several forms.
- Uses `* as [name]` to import all the exported definitions and bundle them in a named object.
- The string following `from` can be a local path or a third-party package.

```
// Import from local module.
import * as adding from './example.js';
// Import from installed module.
import * as promptSync from 'prompt-sync';

console.log(typeof adding); // object

let result = adding.addFive(adding.FIVE);
console.log(result); // 10

console.log(typeof promptSync); // object

// !TypeError: adding.add is not a function
result = adding.add(6, 9);
```

Destructuring Import Syntax

- Offers a second import syntax that resembles object destructuring
- Has the advantage of avoiding the noisy `adding.addFive` object wrapper
- Provides the option of choosing which definitions to import.

```
import { FIVE, addFive } from './example.js';

console.log(typeof addFive); // function

let result = addFive(FIVE);
console.log(result);         // 10
```

Destructuring with an Alias

- Import destructuring uses a different alias syntax than object destructuring.
- It uses the as keyword.
- The advantage of aliasing is that we can assign the name that we want instead of using the given one—maybe, to avoid a naming conflict.

```
import {
    FIVE as five,
    addFive as plusFive
} from './example.js';

console.log(typeof plusFive); // function

let result = plusFive(five);
console.log(result);          // 10
```

Exporting a default

- Each module can have one and only one default export.
- Each module can also export as many additional definitions as we want.

```
export const FIVE = 5;

function add(a, b) {
    return a + b;
}

export function addFive(a) {
    return add(a, FIVE);
}

export default function addSix(a) {
    return add(a, 6);
}
```

Importing a Default

- We can import a default without using the `*` or destructuring. We just give it a name.
- The name doesn't have to match the name of the definition.
- We can import other, non-default definitions at the same time.
- The `prompt-sync` module has only a default export.

```
import addSix from './example.js';
import someName from './example.js';
// import more than the default
import plusSix, { addFive } from './example.js';

import createPrompt from 'prompt-sync';
import eslint from 'eslint';

console.log(typeof addSix);       // function
console.log(typeof someName);     // function
console.log(typeof plusSix);      // function
console.log(typeof addFive);      // function
console.log(typeof createPrompt); // function
console.log(typeof eslint);       // object
```

export {}

- The `export` keyword works with an object syntax.
- Again, the syntax isn't exactly that of an object, because we can alias with the `as` keyword.

```
const FIVE = 5;

function add(a, b) {
    return a + b;
}

function addFive(a) {
    return add(a, FIVE);
}

function addSix(a) {
    return add(a, 6);
}

export {
    FIVE as five,
    addFive as plusFive,
    addSix
};
```


Break

Modular Refactor for Tic-Tac-Toe

Suggested Time:

10 Minutes

Doggie Day Care with Modules

Suggested Time:

45 Minutes

Recap

Learning Outcomes

By the end of this lesson, you will be able to:

01 Export a definition.

02 Import a definition.

03 Export a default definition or expression.

04 Import a default definition or expression.

05 Use aliases for both exports and imports.

Questions?