Background Story

It was mid-afternoon in Bonifacio Global City, and the startup office buzzed like a busy jeepney terminal 🚌. Odessa stared at her screen: dozens of functions—schedule checker, date formatter, email sender—all mixed in one giant `app.js`. Every change felt risky. One small tweak would break something else, and her teammates were already groaning over merge conflicts.

“Odessa, we need some orden!” said Kuya Ronnie, peering over her shoulder. He tapped the file tree. “Let’s split this chaos into modules. Export your helpers, import where you need them. Clean folder structure—like organizing your books in a shelf.”

Odessa imagined her code in neat barangay rows. She created a `utils/` folder and moved date helpers into `dateUtils.js`. She wrote at the top:

```
// dateUtils.js
export function formatDate(d) {
  // …
}

export function daysBetween(a, b) {
  // …
}
```

Then in `emailScheduler.js` she imported:

```
import { formatDate, daysBetween } from "./utils/dateUtils.js";
```

She felt that satisfying snap—like clearing her browser cache and seeing everything load faster. Next, Kuya Ronnie taught her about default exports:

```
// config.js
const config = { timezone: "Asia/Manila", retries: 3 };
export default config;
```

And how to import it:

```
import config from "./config.js";
```

Odessa updated her scheduler to use `config.timezone`. Magic! No more hard-coded strings.

Finally, Kuya Ronnie showed her a “barrel” file—`index.js` inside `utils/` that re-exports everything:

```
// utils/index.js
export * from "./dateUtils.js";
export * from "./stringUtils.js";
```

Now she could import helpers with one line:

```
import { formatDate, capitalize } from "./utils/index.js";
```

Odessa's workspace looked cleaner than her favorite bookshelf at home 📚. Pull requests became smaller, reviews faster. Her codebase transformed from tangled spaghetti into neatly cut pancit bihon.

That evening, as the Makati skyline turned purple, Odessa committed her changes and pushed to GitHub. She felt proud—her app.js was down from 800 lines to just the entry point, and each module held a clear responsibility. Tomorrow, she'd learn about bundlers like Webpack and Rollup—tools that would package her modules for production. But for now, she basked in the clarity that modular code brings.

Her journey from chaos to clarity had begun. She was dividing and conquering her code one module at a time—just like her dream startup would conquer the fintech world.

Theory & Lecture Content

Modern JavaScript (ES6+) includes a native module system. Key topics:

1. Named Exports & Imports
2. Default Exports & Imports
3. "Barrel" Files (Re-exporting)
4. Introduction to Module Bundlers

1. Named Exports & Imports

Exporting multiple items:

```
// utils/mathUtils.js
export function add(a, b) {
  return a + b;
}
export function subtract(a, b) {
  return a - b;
}
```

Import only what you need:

```
// calculator.js
import { add, subtract } from "./utils/mathUtils.js";
```

```
console.log(add(5, 3)); // 8
console.log(subtract(5, 3)); // 2
```

2. Default Exports & Imports

Each file can have one default export:

```
// config.js
const config = { retries: 3, timeout: 5000 };
export default config;
```

Import without braces:

```
// index.js
import config from "./config.js";
console.log(config.retries); // 3
```

3. Barrel Files (Re-export)

Group multiple modules into one:

```
// utils/index.js
export * from "./mathUtils.js";
export { default as config } from "../config.js";
```

Then import from the barrel:

```
import { add, config } from "./utils/index.js";
```

4. Module Bundlers Intro

Browsers support `<script type="module">`, but older environments need bundlers like Webpack or Rollup. Bundlers:

- Resolve `import`/`export` statements
- Combine modules into single or few files
- Optimize (minify, tree-shake) for production

Typical workflow:

1. Write ES6 modules in `src/`
2. Configure bundler (`webpack.config.js` or `rollup.config.js`)
3. Run `npm run build`

4. Serve bundled code in `dist/bundle.js`

Reference:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

---

Exercises

Exercise 1: Split & Import Named Exports

Problem Statement
Create `utils/mathUtils.js` with named exports `add` and `multiply`, then import and use them in `calculator.js`.

TODOs

- In `utils/mathUtils.js`, export `add(a, b)` and `multiply(a, b)`.
- In `calculator.js`, import both functions and export a `calculate(a, b)` that returns an object `{ sum, product }`.

Starter Code (utils/mathUtils.js)

```
// TODO: export named functions

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

function multiply(a, b) {
  return a * b;
}
```

Starter Code (calculator.js)

```
// TODO: import functions and implement calculate

function calculate(a, b) {
  // use add and multiply
}

module.exports = { calculate };
```

Full Solution (utils/mathUtils.js)

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

```
export function multiply(a, b) {
  return a * b;
}
```

Full Solution (calculator.js)

```
import { add, multiply } from "./utils/mathUtils.js";

export function calculate(a, b) {
  return {
    sum: add(a, b),
    product: multiply(a, b),
  };
}
```

Exercise 2: Default Export & Import

Problem Statement
In `config.js`, default-export an object `{ env, version }`. In `app.js`, import it and export a function `getConfig()` that returns the config object.

TODOs

- In `config.js`, write and default-export the config object.
- In `app.js`, import default export and implement `getConfig()`.

Starter Code (config.js)

```
// TODO: create config object and export default

const config = {
  env: "development",
  version: "1.0.0",
};
```

Starter Code (app.js)

```
// TODO: import default config and implement getConfig

function getConfig() {
  // return config
}

module.exports = { getConfig };
```

Full Solution (config.js)

```
const config = {
  env: "development",
  version: "1.0.0",
};

export default config;
```

Full Solution (app.js)

```
import config from "./config.js";

export function getConfig() {
  return config;
}
```

Exercise 3: Barrel File & Single Import

Problem Statement
Create a barrel file `utils/index.js` that re-exports everything from `mathUtils.js` and `stringUtils.js`. Then in `main.js`, import `add` and `capitalize` from the barrel and export `runAll(a, b, str)` which returns `{ sum, capitalized }`.

TODOs

- In `utils/index.js`, re-export named exports from both modules.
- In `main.js`, import `add` and `capitalize` from `./utils/index.js` and implement `runAll()`.

Starter Code (utils/stringUtils.js)

```
export function capitalize(s) {
  if (!s) return "";
  return s[0].toUpperCase() + s.slice(1).toLowerCase();
}
```

Starter Code (utils/index.js)

```
// TODO: re-export modules
```

Starter Code (main.js)

```

```

```
// TODO: import add and capitalize; implement runAll

function runAll(a, b, str) {
  // return { sum, capitalized }
}

module.exports = { runAll };
```

Full Solution (utils/index.js)

```
export * from "./mathUtils.js";
export * from "./stringUtils.js";
```

Full Solution (main.js)

```
import { add, capitalize } from "./utils/index.js";

export function runAll(a, b, str) {
  return {
    sum: add(a, b),
    capitalized: capitalize(str),
  };
}
```

Test Cases

Create these Jest test files:

calculator.test.js

```
import { calculate } from "./calculator.js";

test("calculate returns sum and product", () => {
  expect(calculate(3, 4)).toEqual({ sum: 7, product: 12 });
  expect(calculate(-1, 5)).toEqual({ sum: 4, product: -5 });
});
```

app.test.js

```
import { getConfig } from "./app.js";

test("getConfig returns default config", () => {
  const cfg = getConfig();
```

```
  expect(cfg).toHaveProperty("env", "development");
  expect(cfg).toHaveProperty("version", "1.0.0");
});
```

main.test.js

```
import { runAll } from "./main.js";

test("runAll returns correct sum and capitalized string", () => {
  const result = runAll(2, 3, "odessa");
  expect(result).toEqual({ sum: 5, capitalized: "Odessa" });
});
```

Closing Story

As the last test passed on her screen, Odessa leaned back and looked around the now-tidy file explorer. Modules in their neat folders felt like color-coded books on a library shelf. Pull requests were smaller, conflicts minimal, and code reviews swift. She smiled at Kuya Ronnie's Slack message: "Nice work, Ods! Next up: bundler magic—Webpack & Rollup!"

In her mind's eye, she saw a production build pipeline, bundling her modules into optimized assets, ready to deploy to Vercel or Netlify. She pictured her startup's codebase growing sustainably—each feature in its own module, easy to test and maintain.

Divide and conquer wasn't just a strategy; it was her new mindset. Tomorrow, she'd learn how to package these modules for the real world and ship production-ready code. But tonight, she savored this victory: clarity born from clean exports and imports. Her code was no longer chaos—it was a symphony of modules, each playing its part. 🎶🚀