In JavaScript, you can use **ES6 modules** (import and export) to share classes, objects, and methods between different files and across parent-child relationships. Below are examples demonstrating different scenarios.

**1. Exporting and Importing a Class**

If you want to define a class in one file and use it in another:

**File: Person.js (Exporting a Class)**

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);

}

}

// Exporting the class

export default Person;

**File: main.js (Importing the Class)**

import Person from './Person.js'; // Importing default export

const person1 = new Person('Alice', 25);

person1.greet();

## ****Common Issues & Fixes****

### ❌ ****Error: Cannot use import statement outside a module****

✔ **Fix:** Ensure "type": "module" is in package.json or use the .mjs extension.

**2. Exporting Multiple Classes or Objects**

You can export multiple elements using **named exports**.

**File: utils.js**

export class Animal {

constructor(species) {

this.species = species;

}

sound() {

console.log(`${this.species} makes a sound`);

}

}

export const PI = 3.14;

export function add(a, b) {

return a + b;

}

**File: main.js (Importing Multiple Elements)**

import { Animal, PI, add } from './utils.js';

const dog = new Animal('Dog');

dog.sound();

console.log(`Value of PI: ${PI}`);

console.log(`Addition result: ${add(5, 10)}`);

**3. Exporting and Importing Methods from an Object**

Instead of classes, you can also export objects containing multiple methods.

**File: mathOperations.js**

const mathOps = {

multiply(x, y) {

return x \* y;

},

divide(x, y) {

return x / y;

}

};

export default mathOps;

**File: main.js**

import mathOps from './mathOperations.js';

console.log(mathOps.multiply(4, 5));

console.log(mathOps.divide(10, 2));

**4. Inheritance between Classes in Different Files**

You can extend a class (inheritance) and use super() to call the parent constructor.

**File: Vehicle.js**

export class Vehicle {

constructor(type) {

this.type = type;

}

describe() {

console.log(`This is a ${this.type}`);

}

}

**File: Car.js**

import { Vehicle } from './Vehicle.js';

export class Car extends Vehicle {

constructor(brand, model) {

super('Car'); // Calling the parent class constructor

this.brand = brand;

this.model = model;

}

details() {

console.log(`${this.brand} ${this.model} is a type of ${this.type}`);

}

}

**File: main.js**

import { Car } from './Car.js';

const myCar = new Car('Toyota', 'Corolla');

myCar.describe();

myCar.details();

**5. Importing everything (\* as alias)**

If you want to import everything from a module:

**File: helpers.js**

export function greet(name) {

return `Hello, ${name}!`;

}

export const version = '1.0.0';

**File: main.js**

import \* as Helpers from './helpers.js';

console.log(Helpers.greet('John'));

console.log(`Version: ${Helpers.version}`);

**6. Default vs Named Exports**

You can mix **default exports** (one per file) with **named exports**.

**File: shapes.js**

export default class Circle {

constructor(radius) {

this.radius = radius;

}

area() {

return Math.PI \* this.radius \* this.radius;

}

}

export function squareArea(side) {

return side \* side;

}

**File: main.js**

import Circle, { squareArea } from './shapes.js';

const circle = new Circle(5);

console.log(`Circle area: ${circle.area()}`);

console.log(`Square area: ${squareArea(4)}`);

**Summary**

| **Export Type** | **Import Syntax** |
| --- | --- |
| export default ClassName; | import ClassName from './file.js'; |
| export { function1, variable }; | import { function1, variable } from './file.js'; |
| export function myFunc() {} | import { myFunc } from './file.js'; |
| export const obj = { key: value }; | import { obj } from './file.js'; |
| export \* from './module.js'; | import \* as alias from './module.js'; |