02-Node-Module-System/05-events.md

⚡ Events & EventEmitter

🎯 Understanding Events

Core Concept in Node.js

Events signal that something happened in your application

🔔 What is an Event?

An **event** is a signal that something happened in your application.

Examples

- 🌐 Web server received a request
- 📁 File finished reading
- ⏰ Timer completed
- 🔌 Connection established

Response

We need to **listen** for events and **react** accordingly!

📚 Events Module

EventEmitter Class

```
const EventEmitter = require('events');
```

Convention: Capital letters = Class

A class has:

- Properties (data)
- Methods (functions)

  📖 Events Module Docs

🎬 Creating an EventEmitter

Basic Usage

```
const EventEmitter = require('events');
const emitter = new EventEmitter();  // Create object

// Register a listener
emitter.on('messageLogged', function() {
    console.log('Listener called');
});

// Emit (raise) an event
emitter.emit('messageLogged');
```

Output:

```
Listener called
```

⚠️ Order Matters!

Listener MUST Be Registered First

```
// ❌ WRONG: Emitting before registering listener
emitter.emit('messageLogged');

emitter.on('messageLogged', function() {
    console.log('Listener called');
});
// Nothing happens!
```

```
// ✅ CORRECT: Register listener first
emitter.on('messageLogged', function() {
    console.log('Listener called');
});

emitter.emit('messageLogged');
// Listener called
```

📦 Event Arguments

Sending Data with Events

Sometimes you need to send data along with the event (like ID, URL, etc.)

```
const EventEmitter = require('events');
const emitter = new EventEmitter();

// Listener receives the argument
emitter.on('messageLogged', function(arg) {
    console.log('Listener called', arg);
});

// Emit with data (object recommended for multiple values)
emitter.emit('messageLogged', { id: 1, url: 'http://...' });
```

Output:

```
milan@les2~ node eventargvb.js
Listener called { id: 1, url: 'http://...' }
milan@les2~
```

Best Practice

Use an **object** for multiple values instead of separate parameters.

🏗️ Extending EventEmitter

Real-World Pattern

Let's build a Logger class that emits events.

Problem: Two separate EventEmitter instances don't communicate!

```
// app.js
const EventEmitter = require('events');
const emitter = new EventEmitter();

emitter.on('messageLogged', (arg) => {
    console.log('Listener called', arg);
});

const log = require('./logger');
log('message'); // Only shows 'message', no event!
```

```
// logger.js
const EventEmitter = require('events');
const emitter = new EventEmitter();  // Different instance!

function log(message) {
    console.log(message);
    emitter.emit('messageLogged', {id: 1, url: 'http//...'});
}

module.exports = log;
```

Problem: The emitters in `app.js` and `logger.js` are different objects!

✅ Solution: Create a Custom Class

Using ES6 Classes

logger.js:

```
const EventEmitter = require('events');

// ES6 Class (PascalCase naming convention)
class Logger extends EventEmitter {
    log(message) {
        // Send HTTP request
        console.log(message);

        // Raise event (use 'this' to refer to current instance)
        this.emit('messageLogged', {id: 1, url: 'http://...'});
    }
}

module.exports = Logger;
```

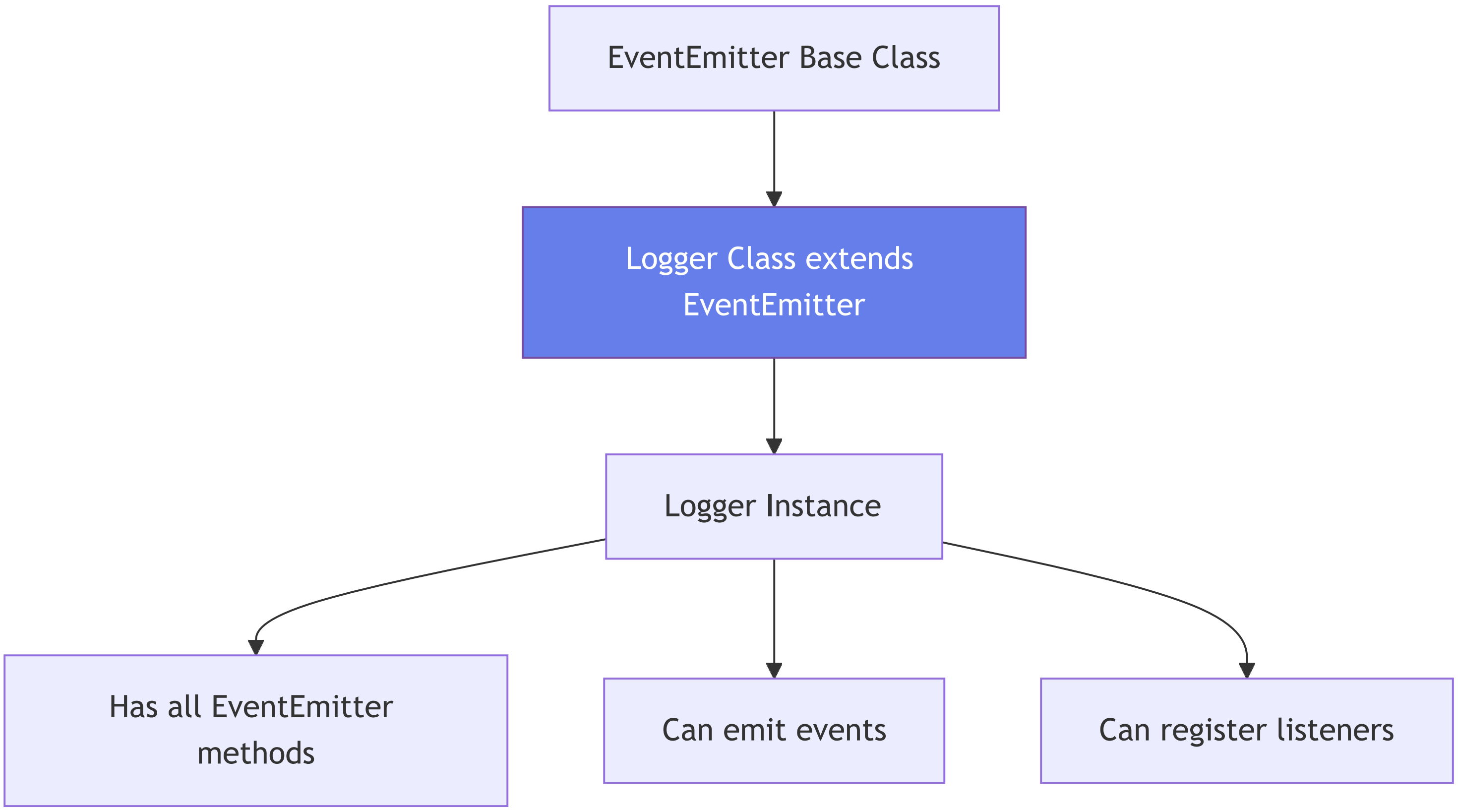
app.js:

```
const Logger = require('./logger');
const logger = new Logger();

// Register listener
logger.on('messageLogged', (arg) => {
    console.log('Listener called', arg);
});

// Call the method
logger.log('message');
```

Output:

```
milan@les2~ node app.js
message
Listener called { id: 1, url: 'http//...' }
milan@les2~
```

🎨 How It Works



🔑 Key Concepts

Understanding the Pattern

Concept	Explanation
EventEmitter	Base class for event handling
extends	Inherits all functionality from EventEmitter
emit()	Raises (triggers) an event
on()	Registers a listener (event handler)
this	Refers to the current instance

💡 Best Practices

DO ✅

- Use PascalCase for class names (Logger, not logger)
- Extend EventEmitter for event-driven classes
- Register listeners before emitting events
- Use objects for event data with multiple properties
- Use arrow functions `() =>` for concise listeners

DON'T ❌

- Don't create separate EventEmitter instances
- Don't emit events before registering listeners
- Don't use too many arguments (use an object instead)
- Don't forget `this` when emitting from a class

🧪 Practice Exercise

Build a Ticket System

Create a `TicketSystem` class that:

1. Extends EventEmitter
2. Has a `newTicket(customer, issue)` method
3. Emits a ‘ticketCreated’ event with ticket data
4. Listens for the event in your main app
5. Logs ticket information when event fires

🏠 Course Home | 📘 Chapter 2 Home

← Previous: Built-in Modules | Next: HTTP Module →