Event-Driven Programming - Complete Notes

⚙️ What is Event-Driven Programming?

Event-Driven Programming is a programming paradigm where the flow of the program is determined by **events** rather than following a sequential, top-to-bottom execution.

In simple words:

- The program waits for events to happen
- When an event occurs, the program reacts by executing specific code
- Instead of running line by line, it responds to actions/triggers

---

🎯 What is an Event?

An **event** is an action or occurrence that happens in the system.

Examples of Events:

- 🖱️ User clicks a button
- ⌨️ User types in a text field
- 📡 Data arrives from a server
- ⏰ A timer completes
- 📁 A file finishes uploading
- 🔔 A notification is received
- 🌐 HTTP request received
- ✅ Database query completes

🔄 Traditional vs Event-Driven Programming

Traditional (Sequential) Programming:

Step 1: Do this

Step 2: Do that

Step 3: Do something else

(Program follows fixed order, waits for each step to complete)

Example:

```
console.log("Step 1");

console.log("Step 2");
```

```
console.log("Step 3");
// Always executes in this order
```

Event-Driven Programming:

Wait for something to happen...

When button is clicked → Do this

When data arrives → Do that

When timer ends → Do something else

(Program reacts to events as they occur)

Example:

```
button.addEventListener('click', () => {
  console.log("Button clicked!");
});
// Code only runs when event happens
```

🔑 Core Concepts of Event-Driven Programming

1. Event Source (Event Emitter)

Objects or elements that generate events

Examples: buttons, timers, network requests, file operations

2. Event Listeners (Event Handlers)

Functions that "listen" for specific events and execute code when the event occurs

3. Event Loop

A mechanism that continuously checks for events and executes corresponding handlers

4. Callbacks

Functions passed as arguments that get executed when an event occurs

---

🌐 How JavaScript and Node.js are Event-Driven

JavaScript and Node.js are **single-threaded** but can handle multiple operations through its **event-driven, non-blocking architecture**.

It uses an **event loop** to handle asynchronous operations efficiently.

How It Works:

1. **Synchronous code** executes line by line on the call stack

2. **Asynchronous operations** (events, timers, API calls) are handled by Web APIs
3. When event completes, **callback** is added to the callback queue
4. **Event Loop** checks if call stack is empty
5. If empty, it moves callback from queue to call stack for execution

Summary:

This is how JS and Node.js are event-driven programming languages. Based on the request/event, we execute the code - not just like traditional systems where we execute the code line by line and wait for the asynchronous task.

If the asynchronous task is there, we give it to the background and we execute the next task, and after the event completes, we handle it.

✅ **Advantages of Event-Driven Programming**

✅ **Non-blocking** - Program doesn't wait, can handle multiple operations

✅ **Responsive** - Perfect for user interfaces and real-time apps

✅ **Efficient** - Resources used only when events occur

✅ **Scalable** - Can handle many concurrent operations

✅ **Flexible** - Easy to add/remove event handlers

✅ **Perfect for I/O operations** - File system, network, databases

---

❌ **Disadvantages of Event-Driven Programming**

❌ **Complex Flow** - Hard to trace program execution

❌ **Debugging** - Difficult to debug asynchronous code

❌ **Callback Hell** - Nested callbacks can get messy

❌ **Error Handling** - Harder to catch and handle errors

❌ **Race Conditions** - Events might fire in unexpected order

---

📊 **Event-Driven vs Sequential Programming**

Feature	Sequential	Event-Driven
Execution	Top to bottom	Based on events
Flow Control	Predictable	Unpredictable

Feature	Sequential	Event-Driven
Blocking	Can block execution	Non-blocking
Concurrency	One task at a time	Multiple tasks
Use Case	Scripts, calculations	UI, real-time apps
Performance	Slower for I/O	Faster for I/O
Examples	Batch processing	Web servers, chat apps

End of Notes