Below is a **clean, simplified, internally accurate, NoteLM-ready version** of **Error Handling in JavaScript**, written in the **same format, depth, and clarity** as your previous topics (Events, Custom Events, Bubbling, Capturing).

I've:

- Removed noise
- Fixed technical inaccuracies
- Added **internal JS engine behavior**
- Kept **examples minimal but powerful**
- Made it **exam + interview ready**

⚠️ Error Handling in JavaScript

(Concept · Internal Working · Types · Handling Mechanisms · Examples)

1️⃣ What is an Error in JavaScript?

An **error** is an abnormal situation that **disrupts the normal execution flow** of a JavaScript program.

When an error occurs:

- JavaScript **stops execution in the current execution context**
- Remaining code in the same call stack **does not run**
- Other independent tasks (event loop callbacks) may still run

📌 Errors directly affect **program reliability and user experience**

2️⃣ Why Error Handling is Required?

Without error handling:

- Application crashes
- UI freezes
- Unexpected behaviour
- Poor user experience

With proper error handling:

- Application becomes **robust**
- Failures are **controlled**
- Errors are **reported gracefully**

3️⃣ Types of Errors in JavaScript

🔹 1. Syntax Errors (Parse-time Errors)

- Occur **before execution**

- Detected while **parsing the code**
- ❌ Cannot be caught using `try...catch`

Example

```
js

console.log("Hello";
```

❌ Missing closing parenthesis → **SyntaxError**

📌 **Script will not execute at all**

🔹 2. Runtime Errors (Exceptions)

- Occur **during execution**
- Syntax is correct, but execution fails
- ✅ Can be caught using `try...catch`

Example

```
js

console.log(notDefinedVar);
```

➡️ `ReferenceError: notDefinedVar is not defined`

🔹 3. Logical Errors

- No error thrown by JS engine
- Program runs but **produces wrong output**
- Most dangerous and hardest to debug

Example

```
js

let result = 10 / 0;
```

✔️ No exception

❌ Logical flaw (`Infinity`)

4️⃣ Internal Working: How JavaScript Handles Errors 🧠

1. JS engine executes code line by line
2. If an error occurs:
    - Execution of **current call stack stops**

- Error object is created

3. Engine searches for:
   - Nearest `try...catch`
4. If not found:
   - Error propagates to **global scope**
   - Program crashes

5️⃣ What is Error Handling?

Error handling means:

> Detecting errors and preventing the program from crashing by handling them gracefully.

JavaScript provides **4 major mechanisms**:

1. `try...catch...finally`
2. `throw`
3. `window.onerror`
4. Custom Errors (`Error`, `CustomError`)

6️⃣ try...catch...finally

🔹 Syntax

```
js

try {
  // risky code
} catch (error) {
  // handle error
} finally {
  // always executes
}
```

📌 Notes:

- `try` → code that may fail
- `catch` → executes only if error occurs
- `finally` → always executes (optional)

🔹 Example: Runtime Error Handling

```
html

<script>
try {
```

```
  myFunction(); // not defined
} catch (e) {
  alert("Error caught: " + e.message);
} finally {
  alert("Cleanup done");
}
</script>
```

✔️ App does not crash

✔️ Error handled gracefully

🔹 Important Rules

- ❌ Syntax errors cannot be caught
- ✔️ Only runtime errors are catchable
- ✔️ One `try` → one `catch` or `finally` (or both)

7️⃣ The `throw` Statement

Used to **manually generate errors**.

🔹 Why use `throw`?

- Enforce business rules
- Validate inputs
- Create controlled failures

🔹 Example

```
js

function divide(a, b) {
  if (b === 0) {
    throw new Error("Division by zero not allowed");
  }
  return a / b;
}

try {
  divide(10, 0);
} catch (e) {
  console.log(e.message);
}
```

✔️ Custom validation

✔️ Controlled exception

8️⃣ `window.onerror` (Global Error Handler)

- Catches **uncaught errors**
- Last safety net
- Attached to `window` object

🔹 Syntax

```
js

window.onerror = function (message, source, line, column, error) {
  // handle error
};
```

🔹 Example

```
html

<script>
window.onerror = function (msg, url, line) {
  console.log("Message:", msg);
  console.log("File:", url);
  console.log("Line:", line);
};
</script>

<button onclick="undefinedFunction()">Click</button>
```

📌 Useful for:

- Logging
- Error reporting tools
- Production monitoring

9️⃣ Error Handling in HTML Elements

Example: Image load error

```
html

<img src="invalid.jpg" onerror="alert('Image failed to load')" />
```

✔️ Common use case

✔️ Lightweight handling

🔟 The Error Object

When an error occurs, JavaScript creates an **Error object**.

🔹 Properties

Property	Description
`name`	Error type
`message`	Error description

🔹 Example

```js
try {
  throw new Error("Something went wrong");
} catch (e) {
  console.log(e.name);    // Error
  console.log(e.message); // Something went wrong
}
```

1️⃣1️⃣ Built-in Error Types

Error Type	Description
`Error`	Base error
`SyntaxError`	Invalid syntax
`ReferenceError`	Undefined variable
`TypeError`	Invalid type operation
`RangeError`	Value out of range
`URIError`	Invalid URI
`AggregateError`	Multiple errors together
`EvalError`	Deprecated

1️⃣2️⃣ Custom Errors

You can define **your own error types**.

🔹 Example

```js
class ValidationError extends Error {
  constructor(message) {
    super(message);
    this.name = "ValidationError";
  }
}

throw new ValidationError("Invalid input");
```

✔️ Clean architecture

✔️ Meaningful error types

1️⃣3️⃣ Common Mistakes ❌

- Catching syntax errors (not possible)
- Swallowing errors silently
- Overusing `try...catch`
- Not logging errors
- Using strings instead of `Error` objects

1️⃣4️⃣ Exam / Viva Short Answers 🎓

Q: What is runtime error?

👉 Error that occurs during execution.

Q: Can try-catch catch syntax errors?

👉❌ No.

Q: What does finally do?

👉 Executes regardless of error occurrence.

Q: What is window.onerror?

👉 Global error handler.

1️⃣5️⃣ Key Takeaways ✅

✔️ Errors stop execution

✔️ try-catch handles runtime errors

✔️ throw creates custom exceptions

✔️ Error object stores error info

✔️ Proper handling improves reliability

⏭️ What's Next?

Type `next` to continue with:

👉 **Async Error Handling (Promises & async/await)**

👉 **Error propagation & rethrowing**

👉 **Real-world error handling patterns**