



## Task 1-Case Study on Exception Handling

Somya

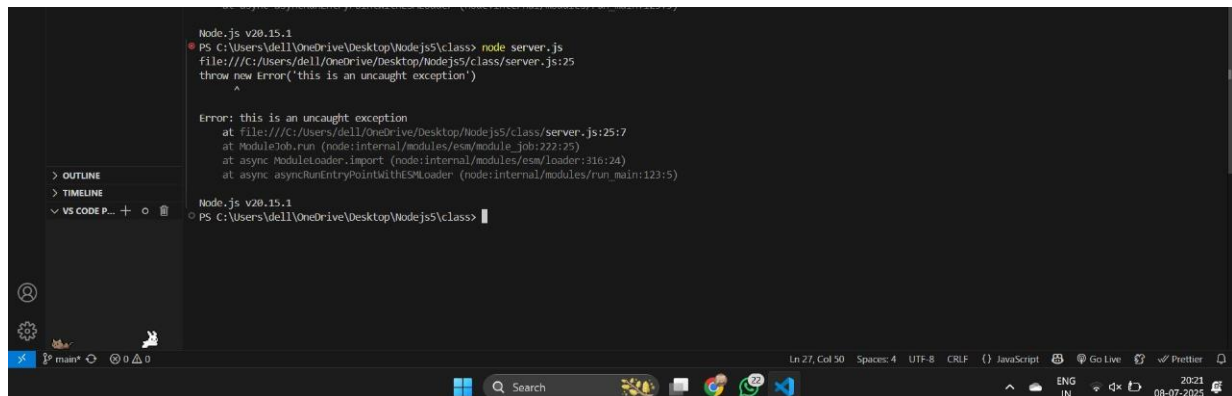
2310991218

G-10

```
//try-catch    block    only    handles  
synchronous exceptions  
  
//promises    handles    asynchronous  
exceptions  
  
throw new Error('this is an uncaught  
exception')
```



```
//try-catch block only handles  
synchronous exceptions  
  
//promises handles asynchronous  
exceptions  
  
throw new Error('this is an uncaught  
exception')  
  
Process.on('uncaughtException' ,  
(err)=>{  
    console.log('uncaught Exception'  
,err.message)  
    Process.exit()  
})
```



```
Node.js v20.15.1
PS C:\Users\dell\OneDrive\Desktop\nodejs\class> node server.js
file:///C:/Users/dell/OneDrive/Desktop/nodejs/class/server.js:25
  throw new Error('this is an uncaught exception')
        ^
Error: this is an uncaught exception
    at file:///C:/Users/dell/OneDrive/Desktop/nodejs/class/server.js:25:7
    at ModuleJob.run (node:internal/modules/esm/module_job:222:25)
    at async ModuleLoader.import (node:internal/modules/esm/loader:316:24)
    at async asynchronousEntryPointWithESMLoader (node:internal/modules/run_main:123:5)

Node.js v20.15.1
PS C:\Users\dell\OneDrive\Desktop\nodejs\class>
```

## Here are answers to some commonly raised doubts among the learners

### Ques 1 : What is the difference between a caught and uncaught exception?

Ans 1 : A caught exception is one that is expected by the program and handled through constructs. Caught exceptions avoid crashes, improve user experience, and make software more reliable by forcing the application to run even when something is wrong. On the other hand, an uncaught exception is one that is not caught by the program. When an exception of this type comes up, it leads to program termination with a stack trace. Uncaught exceptions highlight errors in error-handling design and may lead to system crashes and degrade user experience. Caught vs. uncaught exceptions indicate the maturity and reliability of an application design. Caught exceptions that reflect exercising control and proactive measures, uncaught exceptions are more likely to indicate inadequate testing

### Ques 2 : Why should we exit the process after an uncaught exception?

Ans 2 : When an **uncaught exception** happens in a program, and we didn't plan for it, the program can get confused. It might do stuff like saving the wrong data, freezing, or showing the wrong results. Even worse, we might not even notice that something is broken until later. That is why it's best to exit the process after an uncaught exception appears to take place. By stopping it we can avoid many unwanted problems. It helps keep the program safe.

### Ques 3 : Can you use try...catch to handle errors inside setTimeout? Why or why not?

Ans 3 : Not, If you try to use try-catch **outside** a setTimeout, it won't work. That's because setTimeout runs your code **later**, after the current code is done. So by the time the error actually happens inside the setTimeout, the try-catch has already finished and can't catch anything anymore. The right way to handle errors in this case is to put the try-catch **inside** the setTimeout itself. That way, when the delayed code runs and something goes wrong, the error gets caught and handled properly. It's a common mistake, but once you get how JavaScript handles timing, it makes a lot more sense.

**Ques 4 : What tool can you use in production to restart apps that crash due to uncaught exceptions?**

Ans 4 : Tools like PM2 can be used in production to restart apps that crash due to uncaught exceptions. It behaves like a gaurd. If something goes wrong and the app crashes, PM2 notices and restarts it right away. It is very useful in apps used in real world . your app can crash and come up real quick all by itself i.e automatically , and we don't even have to do anything.