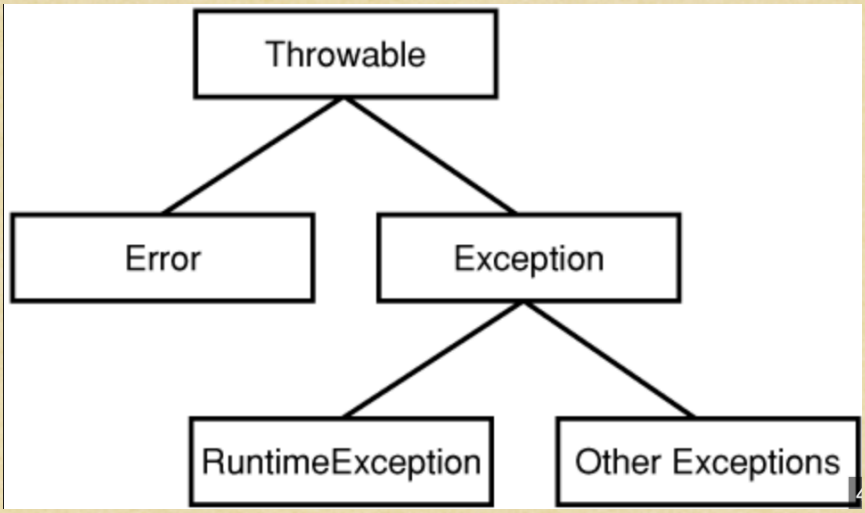
→ Agenda

1) Exception handling
2) Collections.
3) Semaphore code

1) Exception handling:

Whenever something goes wrong, there are 2 things that can happen: throw an Exception or throw an Error.



Exception
- Checked exception
- Non-checked / Runtime exception.

```
class Calculator {

    public int divide (int a, int b) {
        int result = a/b;
        return result;
    }
}
```

a = 5
b = 0 | ⇒ result ⇒ exception

Arithmetic Exception

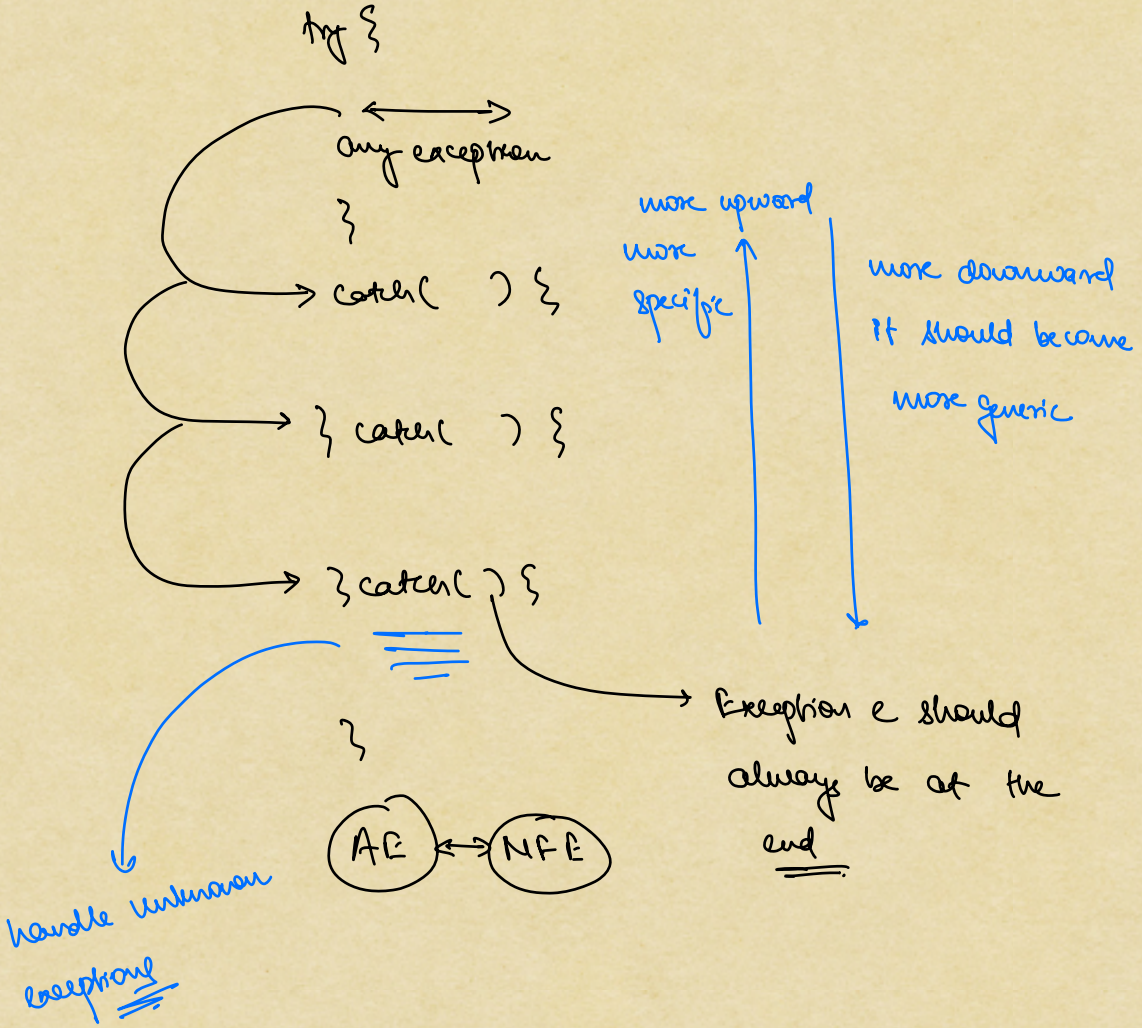
1) How to handle exception ⇒ try | catch | finally

2) Custom exceptions

⇒ How to handle exceptions

try | catch | finally

```
try {
    ==========   → Code that might throw exception
}
catch (ArithmeticException) {
    ==========
}
```



1) Can we have multiple try blocks ⇒ YES

2) try must be followed by catch / finally

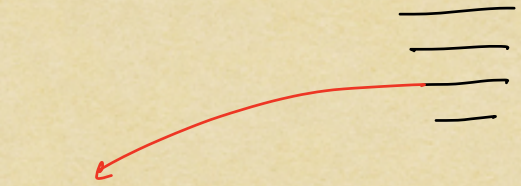
3) Can't write catch / finally without a try

4) Don't write generic exceptions at top.

=> finally

try {

Executor ——————————



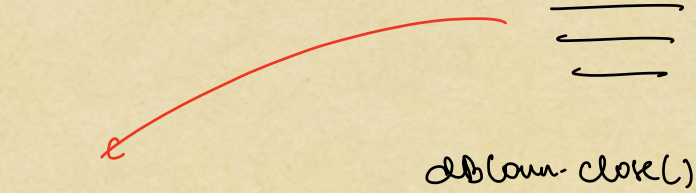
throws an exception

executor.shutdown(); → Cleaning up resources

}

try {

DBConn ——————————



throws an exception

dbConn.close(); → free up resources

}

If any exception occurs, we might not reach the code which will free up the resources.

=> Dont free resources in try block, instead do in finally

* Code inside finally block always executes

⇒ Custom Exceptions

→ find a user by username

→ throw exception ⇒ UsernameNotFoundException

→ find a product by productId

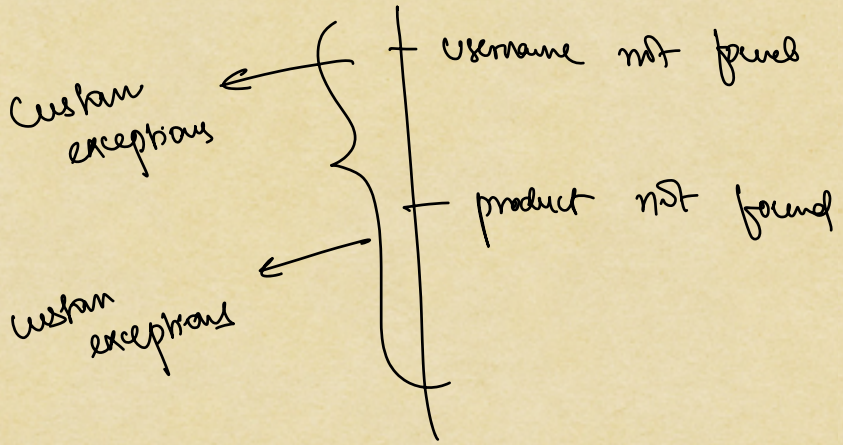
→ throw exception ⇒ ProductNotFoundException

↓

Aspect Oriented programming

↓

aspect ⇒ ControllerAdvice



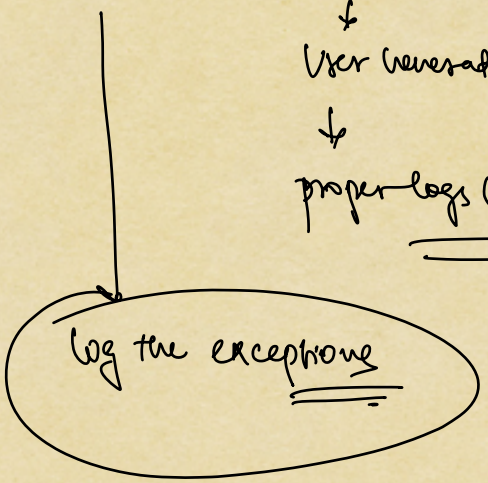
Logs ⇒ Sensitive ⇒ URL

↓

User Generated Content

↓

proper logs (x)



Custom Exceptions

→ Create the class [what exception Exception)

↓

ProductName NotFound Exception

→ extends RuntimeException

→ Create constructors

⇒ Custom Exception

```
if (productName != null) {
        throw new ____________
}
```