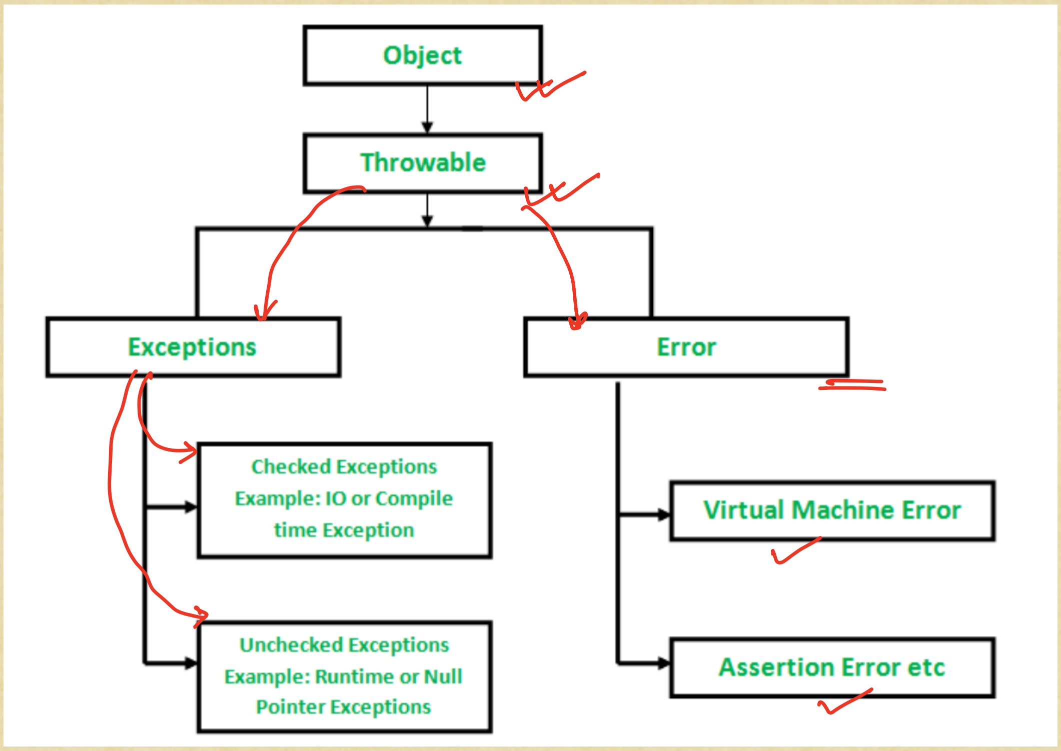
Exception Handling :-

any event that disrupts the normal flow of the code execution is an exception.

$$\begin{tikzcd}
 & \text{recoverable} \arrow[r] & \text{Exception} \\
\text{disruption} \arrow[ur] \arrow[dr] & & \\
 & \text{non-recoverable} \arrow[r] & \text{Error}
\end{tikzcd}$$


Object
Throwable
Exceptions
Error
Checked Exceptions Example: IO or Compile time Exception
Unchecked Exceptions Example: Runtime or Null Pointer Exceptions
Virtual Machine Error
Assertion Error etc

Exceptions and handling them:-

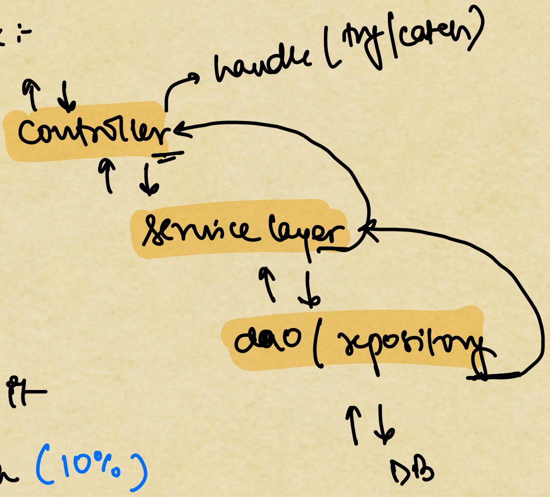
```
Calculator {
        add(x, y)

        sub(x, y)

        mul(x, y)

        division(x, y) => x/y
}
```

division(2,0) $\rightarrow \boxed{2/0} \Rightarrow$ ArithmeticException

* if an exception thrown and it is not handled at some level, it will break the code flow

* any good engineer should be able to think about possible edge cases and exceptions that might occur, and then write code to <u>handle</u> it.

Layered architecture :-

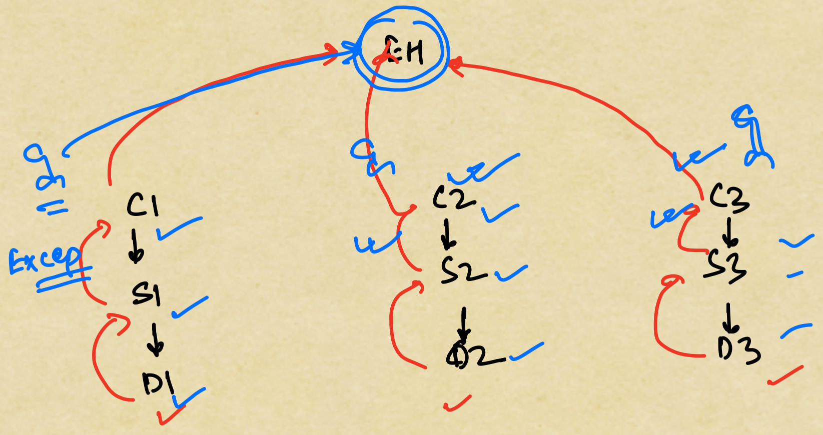


① handle exception where it occurs ⇒ try / catch (10%)

② propagate it above ⇒ throws (90%)

↓

ExceptionHandler ⇒ @ControllerAdvice



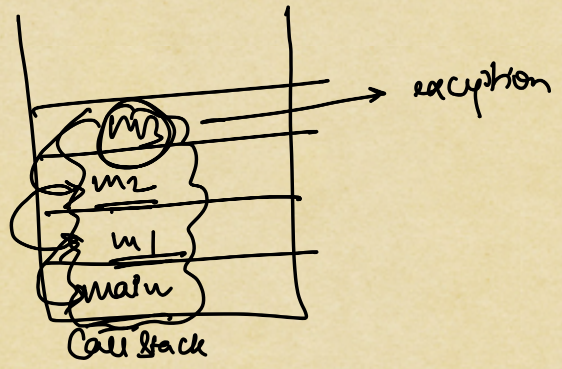
① handle exception where it occurs ⇒ try / catch (10%)

② propagate it above ⇒ throws (90%)

i) try/catch

```
try {
    ______________
    ______________      } code that we want to run
    ______________        and might throw an
    ______________        exception

} catch ( exception ) {
    ______________
    ______________      } actions we need to
    ______________        take after catching an
    ______________        exception

}
```

$$\begin{tikzcd}
\text{psvm} \arrow[dr] & & & \\
& \text{m1} \arrow[dr] & & \\
& & \text{m2} \arrow[dr] & \\
& & & \text{m3}
\end{tikzcd}$$



Should we catch with Exception.java class directly.

Ans ⇒ Yes / NO

NO ⇒ try to be as precise as possible when handling exception but,

Yes ⇒ at some level at topmost level, keep a catch with Exception.java

or, end

⇒ throws

$$\begin{tikzcd}
& \text{throws} \arrow[dl] \arrow[dr] & \\
\text{Checked Exception} \arrow[d] & & \text{Unchecked exception} \arrow[d] \\
\text{Compile} \arrow[d] & & \text{Runtime} \arrow[d] \\
\text{SQLException, IOException} & & \text{NullPointer, Arithmetic, ArrayIndexOutOfB.}
\end{tikzcd}$$

try / catch
finally { → closing up resources, always executes
}

* finalize() → Object ⇒ deprecated
↓
can be overridden

if you are using resources inside your object,
then you close them in finalize();

before GC cleans the object, it calls
finalize() → so that resources get
cleaned up

* final → attributes ⇒ constant (can't change it)
* final → method ⇒ can't override
* final → class ⇒ inherit not possible