11.7-Ducking Exception using throws

Introduction:

The Java **throws** keyword is used to declare an exception in a method's signature. It informs the programmer that an exception **may** occur during the execution of the method. As a result, it encourages programmers to provide appropriate exception-handling code to maintain the normal flow of the program.

Key Points:

- The throws keyword is mainly used to handle **checked exceptions**.
- If an unchecked exception (e.g., NullPointerException) occurs, it is typically considered the programmer's fault for not verifying the code before execution.

Syntax of the throws Keyword:

```
return_type method_name() throws exception_class_name {
    // method code
}
```

Advantages of Using the throws Keyword:

1. **Propagation of Checked Exceptions:** Checked exceptions can be forwarded up the call stack.
2. **Provides Information to the Caller:** The caller of the method is informed about possible exceptions, allowing them to handle it properly.

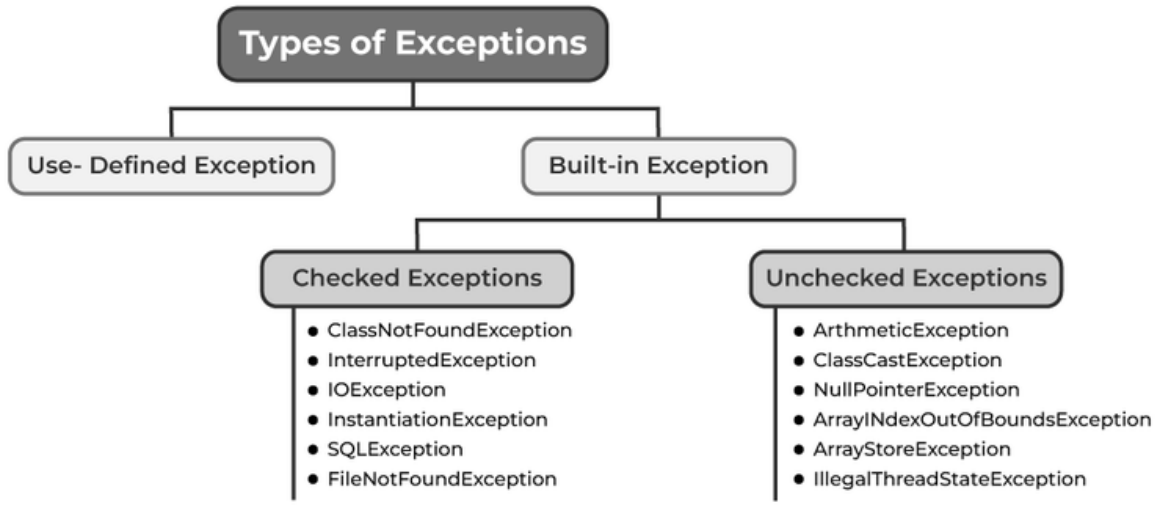
Scenario: Ducking Exceptions

Consider a method **d()** that contains multiple statements, one of which is **critical**. A critical statement is one that might throw an exception. Additionally, two other methods, **e()** and **f()**, each also have a critical statement.

Now, let's assume the main() method calls all of these methods. Since each method contains critical code that might throw an exception, there are two ways to handle this situation:

- **Handle the exception** within each method using a try-catch block.
- **Duck the exception** by not handling it within the method but rather passing it on to the method's caller using the **throws keyword.**

In our case, the main() method will call these critical methods. Each method (e.g., d(), e(), and f()) throws the exception back to the main() method instead of handling it. This is achieved using the throws keyword followed by the exact exception type (or its parent class).



At this point, it is up to the main() method to either handle the exception or duck it further. However, it's considered bad practice to

let the main() method throw the exception without handling it. If the main() method also ducks the exception, the Java Virtual Machine (JVM) will handle it using its **default exception handler**, which is not ideal in real-world coding.

Example:

```
class A {
    public void show() throws ClassNotFoundException {
        System.out.println("In show");
        Class.forName("Calculator"); // Might throw ClassNotFoundException
    }
}

public class Hello {
    public static void main(String[] args) {
        A obj = new A();
        try {
            obj.show();
        } catch (ClassNotFoundException e) {
            System.out.println("Class not found: " + e);
            e.printStackTrace();
        }
    }
}
```

Output:

```
In show
Class not found: java.lang.ClassNotFoundException: Calculator
java.lang.ClassNotFoundException: Calculator
    at java.base/jdk.internal.loader.BuiltinClassLoader.loadClass(BuiltinClassLoader.
    at java.base/jdk.internal.loader.ClassLoaders$AppClassLoader.loadClass(ClassLoade
    at java.base/java.lang.ClassLoader.loadClass(ClassLoader.java:526)
    at java.base/java.lang.Class.forName0(Native Method)
    at java.base/java.lang.Class.forName(Class.java:421)
    at java.base/java.lang.Class.forName(Class.java:412)
    at A.show(MyClass.java:17)
    at MyClass.main(MyClass.java:24)
```

Explanation:

In this example, the A class contains a **show()** method, which calls **Class.forName().** This method might throw a **ClassNotFoundException** (a checked exception). Instead of handling

the exception within show(), it throws it to the main() method using the throws keyword.

The main() method surrounds the show() call with a try-catch block, ensuring the exception is caught and handled properly.

The **e.printStackTrace()** method outputs the stack trace, which shows the method call hierarchy, tracing the path of execution from the origin of the exception to the point where it occurred.

Differences Between throw and throws

Sr. No.	Basis of Differences	throw	throws
1	**Definition**	Used to explicitly throw an exception inside a block of code.	Used in the method signature to declare exceptions that might be thrown during execution.
2	**Usage**	Can propagate unchecked exceptions only.	Can declare both checked and unchecked exceptions but is primarily used for checked exceptions.
3	**Syntax**	Followed by an instance of the exception to be thrown.	Followed by the class names of the exceptions.
4	**Declaration**	Used inside the method body.	Used with the method signature.

Sr. No.	Basis of Differences	throw	throws
5	**Internal Implementation**	Only one exception can be thrown at a time.	Multiple exceptions can be declared using throws. For example: main() throws IOException, SQLException.