Exception Handling

Exception Handling in Java is the process of handling unexpected events (exceptions) that occur during program execution. It prevents the program from crashing and helps manage errors like file not found, invalid input, null object access, arithmetic errors, and more.

Advantages of Exception Handling (Correct + Improved)
✔ 1. Maintains the normal flow of the application
Without exception handling, the program **stops immediately** when an error occurs.
With handling, the program continues smoothly.

✔ 2. Separates error-handling code from normal code
The logic stays clean because error-handling code goes inside catch blocks.

✔ 3. Easy debugging and error identification
Exception messages and stack traces make it easy to find the root cause of errors.

✔ 4. Allows handling of different types of errors separately
Using multiple catch blocks, each type of exception can be handled differently.

✔ 5. Prevents program crashes
Instead of the application terminating abruptly, exceptions are handled gracefully.

✔ 6. Improves application reliability and robustness
Handled exceptions make the program stable even in unexpected situations.

✔ 7. Helps with resource cleanup using finally
finally ensures important resources like files, scanners, and DB connections are always closed.

2 Types of exception

There are mainly two types of exceptions: checked and unchecked.

1. Checked exceptions

1. InterruptExxception
2. FilenotfoundException
3. IOException

2. Un-checked exceptions

1. ArithmeticException
2. NullPointerException
3. NumberformatException
4. ArrayIndexOutofboundException
5. Classcastexception
6. Stringoutofboundexception

Keyword Description

try -- The "try" keyword is used to specify a block where we should place an exception code.

It means we can't use try block alone. The try block must be followed by either catch or finally.

catch -- The "catch" block is used to handle the exception. It must be preceded by try block which means

we can't use catch block alone. It can be followed by finally block later.

finally -- The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not.

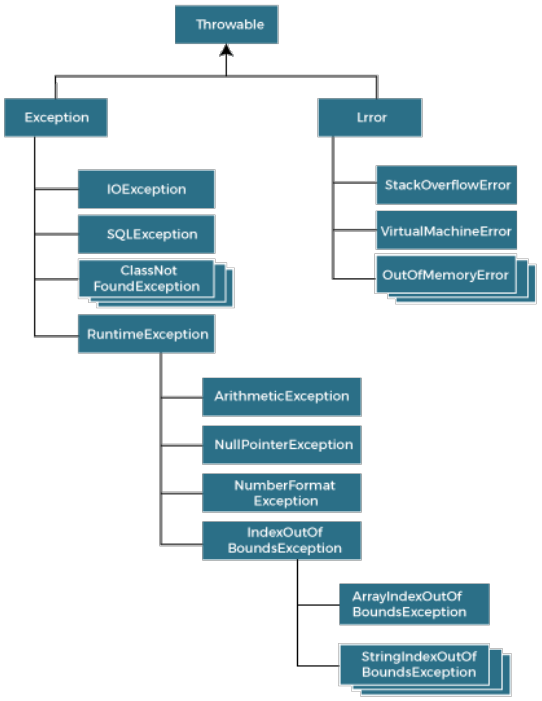
throw -- The "throw" keyword is used to throw an exception.

throws -- The "throws" keyword is used to declare exceptions.
It specifies that there may occur an exception in the method.
It doesn't throw an exception. It is always used with method signature.

Difference b/w throw and throws

Feature	throw	throws
Definition	Used to explicitly throw an exception inside method/block	Used in method signature to declare exceptions that may occur
Where it is used?	Inside a method or block	With method signature
Purpose	To manually throw a single exception	To declare one or more exceptions that the method may throw
Multiple Exceptions ?	Cannot throw multiple exceptions at once	Can declare multiple exceptions (e.g., `throws IOException, SQLException`)
Who handles it?	Handled by surrounding `try-catch` block	Caller of the method must handle it using `try-catch` or declar e further
Type of Exception	Used for both checked & unchecked exceptions	Mostly used for checked exceptions
Execution	Occurs during runtime	Checked at compile time
Example	`throw new ArithmeticException("Error");`	`void readFile() throws IOException {}`

Hierarchy of Java Exception classes



📘 Exception Handling – Important Points for Notes

1. Exception Handling is a mechanism to handle unwanted runtime errors in a graceful manner.
2. It prevents the program from crashing and allows it to continue running.
3. Exceptions occur during runtime (NOT compile time).
4. Java uses a combination of try, catch, finally, throw, and throws to handle exceptions.
5. The parent class of all exceptions is java.lang.Exception.
6. Throwable → Exception + Error
   - Exception → recoverable (e.g., FileNotFound, ArithmeticException)
   - Error → not recoverable (e.g., OutOfMemoryError)
7. try block contains risky code that may throw an exception.
8. catch block handles the exception thrown by try block.
9. finally block always executes whether exception occurs or not.
10. throw keyword is used to throw an exception manually.
11. throws keyword is used in method signature to declare exceptions.
12. Multiple catch blocks allow different handling for different exceptions.
13. Only one catch block executes — the first matching exception type.
14. Catch blocks should be ordered from child exception → parent exception.
15. You cannot write catch without try.
16. You cannot write finally without try, but you can write try + finally without catch.
17. A single try block can have multiple catch blocks.
18. finally is mostly used for closing connections, files, and resources.
19. Custom exceptions can be created using a class that extends Exception.
20. Checked vs Unchecked Exceptions:
    - Checked → compulsory to handle (e.g., IOException)
    - Unchecked → not compulsory (e.g., NullPointerException)
21. Common runtime exceptions include:
ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, NumberFormatException, ClassCastException.
22. Exception handling improves code readability and stability.
23. Exception handling separates error-handling logic from normal business logic.
24. Try block cannot stand alone; it must be followed by either catch or finally.
25. We can also rethrow an exception after catching it (throw e).

📘 Difference Between Exception and Error in Java

⭐ 1. What is an Exception?

An Exception is an unwanted event that occurs during program execution which disrupts the normal flow of the program.

✔ Exceptions are **recoverable**
✔ Can be handled using **try–catch**
✔ Examples:

- ArithmeticException
- NullPointerException
- ArrayIndexOutOfBoundsException
- NumberFormatException
- IOException

2. What is an Error?

An **Error** is a serious problem that occurs **outside the program**, usually related to the **system or JVM**.

❌ Errors are **NOT recoverable**

❌ We cannot handle errors using try–catch

❌ Program usually crashes

✔ Examples:

- OutOfMemoryError
- StackOverflowError
- VirtualMachineError
- NoClassDefFoundError

1. This Keyword

this keyword is used to refer to the **current class object**.

It is mainly used for:

- Differentiating between **instance/global/class level variables** and **local variables** (parameter names).
- Calling other constructors of the same class.
- Calling current class methods.

2. Super Keyword

super keyword is used to refer to the **parent (super) class**.

It is mainly used for:

- Accessing parent class variables
- Calling parent class methods
- Calling parent class constructor

📘 final vs finally vs finalize()

🟩 1. final (Keyword)

✔ final is a keyword in Java
Used to make:
1. **final variable** → value cannot be changed
2. **final method** → cannot be overridden
3. **final class** → cannot be inherited

🟦 2. finally (Block)

✔ finally is a block
✔ Always executes (whether exception occurs or not)
✔ Used to close resources → files, DB connections, sockets

🟧 3. finalize() (Method)

✔ finalize() is a method
✔ Belongs to Object class
✔ Called by Garbage Collector (GC) before destroying an object
✔ Used to clean up unused resources

Aspect	final	finally	finalize()
Definition	final is the keyword and access modifier that is used to apply restrictions on a class, method or variable.	finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.	finalize is the method in Java that is used to perform cleanup processing just before an object is garbage collected.
Applicable to	The final keyword is used with the classes, methods and variables.	Finally block is always related to the try and catch block in exception handling.	The finalize() method is used with the objects.
Functionality	(1) Once declared, a final variable becomes a constant and cannot be modified. (2) The final method cannot be overridden by sub class. (3) The final class cannot be inherited.	(1) finally block runs the important code even if an exception occurs or not. (2) The finally block cleans up all the resources used in a try block	The finalize() method performs the cleaning activities with respect to the object before its destruction.
Execution	The final method is executed only when we call it.	Finally block is executed as soon as the try-catch block is executed. Its execution is not dependant on the exception.	The finalize() method is executed just before the object is destroyed.
Inheritance/Overriding	Prevents method overriding and class inheritance.	Not related to inheritance or overriding.	Can be overridden from the Object class.
Use Case	To create constants, prevent subclassing or method overriding.	To ensure resource cleanup, like closing files, streams, etc., in exception handling.	To perform cleanup activities like memory/resource release before the object is collected.
Control	Gives compile-time control to avoid unintended changes.	Provides runtime control to handle cleanup post-exception or normal flow.	Invoked by JVM (Java Virtual Machine); the programmer cannot call it directly for cleanup.
Example	final int x=10;	Finally{closeResource	protected void finalize