

Q. What is the purpose of exception handling?

=> To handle runtime exceptions.

e.g. NullPointerException

### Q What are the Exception Handling Keywords in Java?

* try, catch, finally, throw, throws

Q. Can we use Object class in catch block?

=> No

Q. Can we use Throwable class in catch block?

=> Yes

Q. Can we use Error class in catch block?

=> Yes

Any Throwable type we can use in catch block.

Q. Can we write down another try catch block in catch block?

=> Yes

Analogy : In govt. org, to catch corrupt officers, anticorruption body(officers), and to catch anticorruption officer another anticorruption body is there

Q. Can we write down another try catch block in finally block?

=> Yes

Q. Can we have nested try block?

=> Yes.

Q. What should be sequence of try catch finally?

Try block first

Then n number of catch blocks

Then we can have finally block.

e.g. you can’t sequence like try then finally and then catch

Q. Can we have multiple catch blocks?

=> Yes

Q. What is the rule while writing multiple catch blocks?

=> first child class and then parent class.

e.g.

catch(ArithmeticException ae){….}

catch(Exception e){..}

So here we specified ArithmeticException(which is child of Exception) first and then Exception class

Q. What is NullPointerException ?

Without creating object, if you call any method then it generates NullPointerException.

Q. What is ArrayIndexOutOfBoundsException ?

=> If index position goes outside the range, then we get ArrayIndexOutOfBoundsException

### What is OutOfMemoryError in Java?

OutOfMemoryError in Java is a subclass of java.lang.VirtualMachineError and it’s thrown by JVM when it ran out of heap memory.

Q. What is ClassCastException?

=> If you do not cast the object properly then you get ClassCastException.

e.g.

Employee

Developer Tester

You converted from Developer to Employee and now you are converting from Employee to Tester, then generates ClassCastException.

Employee e = new Developer();

Tester t=(Tester)e; // this line generates ClassCastException

Q. How to print error messages?

=> 1) hard coded message e.g. System.out.println(“error message”);

2) getMessage() e.g. catch(ArithmeticException ae){ System.out.println(ae.getMessage());

3) printStackTrace() e.g. catch(ArithmeticeException ae){ ae.printStackTrace());

Q. When to use getMessage() and printStackTrace() ?

=> getMessage() – Returns the detail message string of this throwable.

=> printStackTrace() - Prints this throwable and its backtrace to the standard error stream.

Whenever there is a method chaining, in that case we use printStackTrace.

e.g.

**public** **class** Test {

**static** **void** m1()

{

*m2*();

}

**private** **static** **void** m2() {

*m3*();

}

**private** **static** **void** m3() {

**throw** **new** NullPointerException();

}

**public** **static** **void** main(String[] args) {

**try**

{

*m1*();

}

**catch**(NullPointerException n)

{

//System.out.println("Error message");

n.printStackTrace();

}

System.***out***.println("Last message in main");

}

}

Output =>

java.lang.NullPointerException

at Test.m3(Test.java:17)

at Test.m2(Test.java:12)

at Test.m1(Test.java:8)

at Test.main(Test.java:27)

Last message in main

Q. How do you differentiate category of checked and unchecked exception ?

=> Runtime exception or its subclasses and

Error class or its subclasses are called unchecked exceptions.

All other classes exceptions are called checked exceptions

### What is the difference between Checked and Unchecked Exceptions in Java?

1. Checked Exceptions should be handled in the code using try-catch block or else the method should use the throws keyword to let the caller know about the checked exceptions that might be thrown from the method. Unchecked Exceptions are not required to be handled in the program or to mention them in the throws clause of the method.
2. Exception is the superclass of all checked exceptions whereas RuntimeException is the superclass of all unchecked exceptions. Note that RuntimeException is the child class of Exception.
3. Checked exceptions are error scenarios that require to be handled in the code, or else you will get compile time error. For example, if you use FileReader to read a file, it throws FileNotFoundException and we must catch it in the try-catch block or throw it again to the caller method. Unchecked exceptions are mostly caused by poor programming, for example, NullPointerException when invoking a method on an object reference without making sure that it’s not null. For example, I can write a method to remove all the vowels from the string. It’s the caller’s responsibility to make sure not to pass a null string. I might change the method to handle these scenarioEs but ideally, the caller should take care of this.

### What is the difference between the throw and throws keyword in Java?

throws keyword is used with method signature to declare the exceptions that the method might throw whereas throw keyword is used to disrupt the flow of the program and handing over the exception object to runtime to handle it.

### How to write custom exceptions in Java?

By extending any throwable type we can create custom exception.

e.g.

1) class MyException extends ArithmeticException

2) class MyException extends Exception

3) class MyException extends Throwable

Difference between final, finally, finalize

1) final

final can be used with var, method and class

with var - it becomes constant

with method - method cannot be overridden

with class - cannot be extended

2) finally

finally block we use with exception handling

purpose - to close the resources

e.g. file closing, database connection closing

3) finalize

- method from Object class

Finalize is used to perform clean up processing just before object is garbage collected.

Q. Java 7 features multicatch exception handling?

Multicatch exception where we specify more than one exception in a single catch block.

e.g.

catch(NullPointerException | ArithmeticException)

{…..}

Rule :

**try**

{

}**catch**(ArithmeticException | RuntimeException ae)

{

System.***out***.println(ae.getMessage());

}

In a single catch block can’t write child as well as parent class otherwise get following exception:

The exception ArithmeticException is already caught by the alternative RuntimeException

Q Java 7 features try with resources ?

try with resources:

import java.io.\*;

class Main {

public static void main(String[] args) {

String line;

try(BufferedReader br = new BufferedReader(new FileReader("test.txt"))) {

while ((line = br.readLine()) != null) {

System.out.println("Line =>"+line);

}

} catch (IOException e) {

System.out.println("IOException in try block =>" + e.getMessage());

}

}

}

import java.io.\*;

import java.util.\*;

class Main {

public static void main(String[] args) throws IOException{

try (Scanner scanner = new Scanner(new File("testRead.txt"));

PrintWriter writer = new PrintWriter(new File("testWrite.txt"))) {

while (scanner.hasNext()) {

writer.print(scanner.nextLine());

}

}

}

}

**try** (Scanner scanner = **new** Scanner(**new** File("test.txt"))) { **while** (scanner.hasNext()) { System.out.println(scanner.nextLine()); } } **catch** (FileNotFoundException fnfe) { fnfe.printStackTrace(); }

Rule : If you want to any object in try block, that object must be autoclosable.

Scanner implements AutoCloseable

### Can we have an empty catch block?

As far as the syntax is concern, we can have empty catch block but its bad programming.

In my life I saw termination of employee!!

### Provide some Java Exception Handling Best Practices?

Some of the best practices related to Java Exception Handling are:

* Use Specific Exceptions for ease of debugging.
* Throw Exceptions Early (Fail-Fast) in the program.
* Catch Exceptions late in the program, let the caller handle the exception.
* Use Java 7 ARM feature to make sure resources are closed or use finally block to close them properly.
* Always log exception messages for debugging purposes.
* Use multi-catch block for cleaner close.
* Use custom exceptions to throw a single type of exception from your application API.
* Follow naming convention, always end with Exception.
* Document the Exceptions Thrown by a method using @throws in javadoc.
* Exceptions are costly, so throw it only when it makes sense. Else you can catch them and provide a null or empty response.

Method overriding rules with exception handling:

Class Parent

{

void show(){….}

}

Class Child extends Parent

{

void show(){..}

}

Method overriding rules :

1. As far as the primitive return type is concern, it must be same.
2. Covariant e.g. Parent show() in parent class and Child show() in child class in possible.

If the superclass method is returning superclass object then subclass method may return either superclass object or subclass object.

1. While overriding method access modifier should be same or broader modifier is required. (Exception – with private,final,static overriding is not possible)

In above example while overriding we may go for default or protected or public

1. With unchecked exceptions

We can have any number of exceptions in superclass method or subclass method

e.g.

Parent show() is throwing ArithmeticException &

Child show() is throwing NullPointerException and ClassCastException.

1. With checked exceptions

While overriding

1. Can use same class or
2. Subclass or
3. No class

e.g.

Class Parent

{

void show()throws Exception{….}

}

Class Child extends Parent

{

void show()throws Throwable{..} // illegal

}

Following throws are possible while overriding:

a)throws Exception

b) not writing any exception after method

c) throws IOException

Q. Where will you store your exception messages?

=> Logger