Topic Name :🡪 Exception Handling

Q) Can we throw exception to main method ?

* Yes, It is valid to add throws with main method.
* The rules for exception for main method is same as it is for other methods.
* When throws is added to method declaration then it is the responsibility of the calling method to handle exception. Here JVM calls the main method, so thrown exception is handled by JVM.
* When exception is thrown by main() method, although there will be no compile time error but java runtime terminates the program and print the exception message on console.

Q) Can we create user defined exception in Java if so why?

* Yes, we can create user defined exception in java.
* Sometimes it is required to develop meaningful exceptions based on application requirements.
* For example suppose you have one savings account in SBI Bank and you have 50000 in your account. Suppose you attempted to withdraw 60000 from your account. In java you can handle. You need to display some error message related to insufficient fund.
* As there is no such inbuilt exception in java or relevant exception in java so we create such exception.

**package** com.test;

**public** **class** InsufficientFundException **extends** RuntimeException {

**private** String message;

**public** InsufficientFundException(String message) {

//this.message = message;

**super**(message);

}

}

Q) Write a custom exception class/ user defined exception?

* Create a new class whose name should end with Exception like  in our case insufficientfund exception.
* Extend the class to runtime (unchecked) exception rather than checked exception.
* Create a constructor with a String parameter
* In this constructor, simply call the super constructor and pass the message.

Q) What is class not found exception?

* It is a type of checked /compile time exception
* It occurs when the JVM tries to load a particular class and the specified class cannot be found in the classpath.

Q) How to solve class not found exception?

* It is thrown by the application itself. It is thrown by the methods like Class.forName(), loadClass() and findSystemClass().
* It occurs when classpath is not updated with required JAR files.
* To solve this exception, one should correct the errors first by fixing the dependencies. For Eg:

The following steps should be followed to resolve a ClassNotFoundException in Java:

1. Find out which JAR file contains the problematic Java class. For example, in the case of com.mysql.jdbc.driver, the JAR file that contains it is mysql-connector-java.jar.
2. Check whether this JAR is present in the application classpath. If not, the JAR should be added to the classpath in Java and the application should be recompiled.
3. If that JAR is already present in the classpath, make sure the classpath is not overridden (e.g. by a start-up script). After finding out the exact Java classpath used by the application, the JAR file should be added to it.

To fix the Java exception, the mysql-connector JAR should be included in the application classpath.

Q) What is IO exception?

* It is a type of checked/ compile time exception.
* It happens when there is a failure during reading, writing, and searching file or directory operations.
* It is an exception which programmers use in the code to throw a failure in Input & Output operations.

Q) How to solve class not found exception?

Develop can explicitly handle the exception in **a try-catch-finally block** and print out the root cause of the failure. The developer can take the correct actions to solve this **situation by having additional code** in the catch and finally blocks.

Q) What is subclasses of IO exception?

IOException has 5 subclasses such as:

**FileNotFoundException :**

If the file is not found, this exception is thrown.

**EOF Exception:**

While reading a file, EOF Exception occurs when the end of the file is reached.

**UnsupportedEncodingException** :

If the file has an unsupported encoding, this exception occurs.

**SocketException:**

When the socket connection is closed, SocketException can occurs.

**SSLException**

SSLException happens when the SSL connection is not established.

Q) What is SQL exception?

* It is a type of checked/compile time exception.
* It is most common exception in case of JDBC we can say.
* It can provide information on a **database access error** or other errors.
* It can occur both in the driver and the database.

Q) How to solve SQL exception?

* There is no predefined way to solve this exception.
* The developer need to check the JDBC connection steps carefully to avoid this kind of exception.
* Check DriverManager, Class.for Name, and so on.

Q) What is Out of memory exception?

* It is a type of un-checked/run time exception.
* Whenever you create an object in Java it is stored in the heap area of the JVM.
* If the JVM is not able to allocate memory for the newly created objects an exception named OutOfMemoryError is thrown.
* This usually occurs when we are not closing objects for long time or, trying to act huge amount of data at once.
* The Java Garbage Collector (GC) cannot free up the space required for a new object, which causes a java.lang.OutOfMemoryError

Q) How to solve Out of memory exception?

* The easiest solution to remove such error is to increase the memory size of the JVM heap space.
* Instead of trying to increase the heap size of the JVM and still facing the error, one should look for the memory leakage if any is occurring. To examine the memory leakage in Eclipse, make use of the Eclipse Memory Analyzer (known as MAT that is used to find memory leaks and reduce memory consumption)

Q) What is Null pointer Exception?

* It is a type of un-checked/run time exception.
* NullPointerException is thrown when program attempts to use an object reference that has the null value.

For Eg:

**package** com.test;

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

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

String str = **null**;

System.***out***.println(str.length()); //exception will be occured.

}

}

Output

Exception in thread "main" java.lang.NullPointerException

at com.test.Test.main(Test.java:8)

Q) Why do we need the null value?   
Null is a special value used in Java. It is mainly used to indicate that no value is assigned to a reference variable

Q) How to solve Null pointer exception?

* To avoid the NullPointerException, we must ensure that all the objects are initialized properly, before you use them.
* When we declare a reference variable, we must verify that object is not null, before we request a method or a field from the objects.

Q) Diff bet Error & Exception ?

|  |  |
| --- | --- |
| **Errors** | **Exception** |
| All errors in Java are unchecked. | It can be classified into two categories i.e. checked and unchecked. |
| It occurs at run time. | It occurs at compile time or run time. |
| It belongs to java.lang.Error package. | It belongs to java.lang.Exception package. |
| It is irrecoverable | Exception can be recovered by using the try-catch block. |
| Eg: OutOfMemoryError ,IOError | Eg: NullPointerException , SqlException |

Q) What is difference between classcastexception and classnotfoundexception?

**ClassCasteException**

* Itis a unchecked / runtime exception.
* It occurs when JVM unable to cast an object of one type to another type.
* It can also be thrown when we try to convert the objects of two individual classes that don't have any relationship between them.

**ClassNotfoundException**

* It is a type of checked /compile time exception
* It occurs when the JVM tries to load a particular class and the specified class cannot be found in the classpath.

Q) Can we handle/declare multiple exceptions by using single catch block ?

* Yes, Java allows you to catch multiple type exceptions in a single catch block.
* It was introduced in Java 7 and helps to optimize code & increases efficiency.
* You can use vertical bar (|) **pipe symbol** to separate multiple exceptions in catch block.

Example-

import java.util.Scanner;

public class Test

{

    public static void main(String args[])

    {

        Scanner scn = new Scanner(System.in);

        try

        {

            int n = Integer.parseInt(scn.nextLine());

            if (99%n == 0)

                System.out.println(n + " is a factor of 99");

        }

        catch **(NumberFormatException | ArithmeticException ex)**

        {

            System.out.println("Exception encountered " + ex);

        }

    }

}

 Q ) Exception in overriding ?

* Defining multiple methods with same name and same signature in super class and sub class known as method overriding.
* When Overriding a method there is a chance of having statements which may cause exceptions so we need to handle those inside method.
* Otherwise simply we can give responsibility of handling exceptions to calling method.
* We can give the responsibility of handling exceptions of a method to calling place by using throws keyword in java.

**1)If the superclass method does not declare an exception**

* + If the superclass method does not declare an exception,
  + The subclass overridden method cannot declare the **checked exception** but it can declare **unchecked exception**.

**2)If the superclass method declares an exception**

* + If the superclass method declares an exception,
  + Then subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

Q) What is try with resources?

* In Java, the try-with-resources statement is a try statement that declares one or more resources. The resource is as an object that must be closed after finishing the program.
* The try-with-resources statement ensures that each resource is closed at the end of the statement execution.
* You can pass any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable.
* The following example writes a string into a file. It uses an instance of FileOutputStream to write data into the file. FileOutputStream is a resource that must be closed after the program is finished with it.
* So, in this example, closing of resource is done by itself try.

Q) Try & Catch scenarios?

* There are 6 scenarios for exception handling we can say:

**Possible way to write try catch block**

1.

try {

//not allowed}

2.

try {

//allowed

}

catch(Exception e){

}

3.

try {

//allowed

}

finally{

}

4.

try{

//allowed

}

catch (Exception e){

}

finally{

}

5.

try {

//allowed

}

Catch(ArithmaticException e1){

}

Catch(Exception e){

}

6.

try {

//not allowed

}

Catch(Exception e){

}

Catch(ArithmaticException e1){

}

**Rule:** All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Not allowed – What is the reason?

The most general exception cannot be in the first catch because it will accommodate or handle all exceptions and there will be no chance to reach the second catch of Arithmetic exception.

So it is preferable to handle first the most specific exception rather than the most general exception.

Q) Suppose I have written multiple catch block in try block, and I want to handle the IO Exception and I have written 1st catch block with Exception and second block with IO Exception. Will it work?

Q) What happens if we try to write most general exception i.e (Exception e) before any other specific exception i.e (Arithmetic/other exception)?

* It will throw a compile time error .
* It will show **unreacheable catch block** for the given specific exception as exception is already handled in 1st catch block.

Q) If try/catch blocks have a return statement, even then the finally block executes?

* The answer is yes. Finally block will execute. The only case where it will not execute is when it encounters System.exit ().

Q) If we write System.exit() finally block will execute?

* If we call the System.exit () method explicitly in the finally block then only it will not be executed.
* There are few situations where the finally will not be executed like JVM crash, power failure, software crash and etc.
* Other than these conditions, the finally block will be always executed.

Q)What are global Exception?

In Java, exception handling is done by try, catch blocks but **spring boot** also allows us to provide customized global exception handling where we **need not to add try catch block** everwhere, we can create a separate class for handling exceptions and it also separates the exception handling code from businesss logic code.

Q) Can we extend checked exception instead runtime exception ?

* Yes, we can do so.
* If you want to write a checked exception that is automatically enforced by the Handle or Declare Rule, you need to extend the Exception class

Q)Can we catch errors in try & catch block?

* Yes, you can catch an Error, but you are advised not to do it, since Errors indicate serious problems that a reasonable application should not try to catch.
* JVM throws errors to indicate severe problems from which it can't recover, such as lack of memory and stack overflows.
* Thus, we must have a very, very good reason to catch an error!

Q)How to handle Global Exception?

We have to take use of two annotation to handle global exception in spring boot.

The **@ControllerAdvice** is an annotation, to handle the exceptions globally.

The **@ExceptionHandler** is an annotation used to handle the specific exceptions and sending the custom responses to the client.

Steps to handle Global Exception:

1) Create a class and specify @Controller annotation above the class.

2) Extend the class to runtime exception.

3) Define Exception handler method with @Exception handler annotation in class file

The Controller Advice class to handle the exception globally is given below. We can define any Exception Handler methods in this class file.

package com.tutorialspoint.demo.exception;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice

public class ProductExceptionController {

@ExceptionHandler(value = ProductNotfoundException.class)

public ResponseEntity<Object> exception(ProductNotfoundException exception) {

return new ResponseEntity<>("Product not found", HttpStatus.NOT\_FOUND);

}

}

Q) Difference between throw & throws keyword?

|  |  |
| --- | --- |
| **Throw** | **Throws** |
| Java throw keyword is used inside the function or the block of code. | Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code. |
| The **throw** keyword is used to throw an exception explicitly | The **throws** keyword can be used to declare multiple exceptions, separated by a comma. |
| throw is used within the method. | throws is used with the method signature. |
| We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions | We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException. |
| **throw** keyword cannot propagate checked exceptions. It is only used to propagate the unchecked Exceptions that are not checked using the throws keyword. | **throws** keyword is used to propagate the checked Exceptions only. |

Q) Throw & throws keyword which exception they propogate?

**Throw:** Cannot propagate checked exceptions

It is only used to propagate the unchecked Exceptions that are not checked using the throws keyword.

**Throws**: It is used to propagate the checked Exceptions only

Q) Why throws is declared with method signature?

* The signature lets the method's caller know that an exception may occur as a consequence of the call.
* If the exception does get thrown, the caller must be prepared to do something about it

Q) What is exception chaining? How u handled it ?

* Chained Exceptions allows to relate one exception with another exception, i.e one exception describes cause of another exception.
* **For example**, consider a situation in which a method throws an ArithmeticException because of an attempt to divide by zero but the actual cause of exception was an I/O error which caused the divisor to be zero. So the method will throw only ArithmeticException to the caller. So the caller would not come to know about the actual cause of exception. Chained Exception is used in such type of situations.

They are handled by methods & constructor as follows:

**Constructors** Of Throwable class Which support chained exceptions in java :

1. **Throwable(Throwable cause)** :- Where cause is the exception that causes the current exception.
2. **Throwable(String msg, Throwable cause)** :- Where msg is the exception message and cause is the exception that causes the current exception.

**Methods** Of Throwable class Which support chained exceptions in java :

1. **getCause() method :-** This method returns actual cause of an exception.
2. **initCause(Throwable cause) method** :- This method sets the cause for the calling exception.

Q) Exception propagation ?

In Java, an exception is thrown from the top of the stack, if the exception is not caught it is put in the bottom of the stack, this process continues until it reaches to the bottom of the stack and caught. It is known as exception propagation.

For Eg:

**class** TestExceptionPropagation1{

**void** m1(){

**int** data=50/0;

 }

**void** n(){

 m();

 }

**void** p(){

**try**{

 n();

 }**catch**(Exception e){System.out.println("exception handled");}

 }

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

 TestExceptionPropagation1 obj=**new** TestExceptionPropagation1();

 obj.p();

System.out.println("normal flow...");

  }

}

* In the above example exception occurs in the m1() method where it is not handled, so it is propagated to the next n() method where it is not handled, again it is propagated to the p() method where exception is handled.
* Exception can be handled in any method in call stack either in the main() method, p() method, n() method or m() method.

Q) What is rethrowing exception?

* Sometimes we may need to rethrow an exception in Java.
* If a catch block cannot handle the particular exception it has caught, we can rethrow the exception.
* The rethrow expression causes the originally thrown object to be rethrown.
* Because the exception has already been caught at the scope in which the rethrow expression occurs, it is rethrown out to the next enclosing try block.
* Therefore, it cannot be handled by catch blocks at the scope in which the rethrow expression occurred.

Any catch blocks for the enclosing try block have an opportunity to catch the exception.

SYNTAX-

catch(Exception e) {

   System.out.println("An exception was thrown");

   throw e;

}

Q17) What do printstacktrace do?

* It is a method of Java Throwble class.
* It is used to print the Throwable along with other details like classname and line number where the exception occurred.

Syntax🡪

**public** **void** printStackTrace()

**example:**

**import** java.lang.Throwable;

**public** **class** ThrowablePrintStackTraceExample1 {

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

**try**{

**int** i=4/0;

    }**catch**(Throwable e){

        e.printStackTrace();

        System.err.println("Cause : "+e.getCause());

    }

}

}

Q23) Can we write methods in catch block?

* Yes, if we have a class and we have written try & catch block within it so you can call any method that is accessible inside the catch block from other class.