Exception Handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling.

Hierarchy of Java Exception classes

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error. A hierarchy of Java Exception classes are given below:



### Types of Java Exceptions

There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

## Difference between Checked and Unchecked Exceptions

### 1) Checked Exception

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

### 3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone. |
| 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 important 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 doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. |

Java Exception Handling Example

Let's see an example of Java Exception Handling where we using a try-catch statement to handle the exception.

1. **public** **class** JavaExceptionExample{
2. **public** **static** **void** main(String args[]){
3. **try**{
4. //code that may raise exception
5. **int** data=100/0;
6. }**catch**(ArithmeticException e){System.out.println(e);}
7. //rest code of the program
8. System.out.println("rest of the code...");
9. }
10. }

### 1) A scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

1. **int** a=50/0;//ArithmeticException

### 2) A scenario where NullPointerException occurs

If we have a null value in any [variable](https://www.javatpoint.com/java-variables), performing any operation on the variable throws a NullPointerException.

1. String s=**null**;
2. System.out.println(s.length());//NullPointerException

### 3) A scenario where NumberFormatException occurs

The wrong formatting of any value may occur NumberFormatException. Suppose I have a [string](https://www.javatpoint.com/java-string) variable that has characters, converting this variable into digit will occur NumberFormatException.

1. String s="abc";
2. **int** i=Integer.parseInt(s);//NumberFormatException

### 4) A scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

1. **int** a[]=**new** **int**[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException

Java finally block

**Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java finally block follows try or catch block.



1. **class** TestFinallyBlock{
2. **public** **static** **void** main(String args[]){
3. **try**{
4. **int** data=25/5;
5. System.out.println(data);
6. }
7. **catch**(NullPointerException e){System.out.println(e);}
8. **finally**{System.out.println("finally block is always executed");
9. }
10. System.out.println("rest of the code...");
11. }
12. }
13. **class** TestFinallyBlock1{
14. **public** **static** **void** main(String args[]){
15. **try**{
16. **int** data=25/0;
17. System.out.println(data);
18. }
19. **catch**(NullPointerException e){System.out.println(e);}
20. **finally**{System.out.println("finally block is always executed");}
21. System.out.println("rest of the code...");
22. }
23. }
24. **public** **class** TestFinallyBlock2{
25. **public** **static** **void** main(String args[]){
26. **try**{
27. **int** data=25/0;
28. System.out.println(data);
29. }
30. **catch**(ArithmeticException e){System.out.println(e);}
31. **finally**{System.out.println("finally block is always executed");}
32. System.out.println("rest of the code...");
33. }
34. }

class ThrowExcep

{

    static void fun()

    {

        try

        {

            throw new NullPointerException("demo");

        }

        catch(NullPointerException e)

        {

            System.out.println("Caught inside fun().");

            throw e; // rethrowing the exception

        }

    }

    public static void main(String args[])

    {

        try

        {

            fun();

        }

        catch(NullPointerException e)

        {

            System.out.println("Caught in main.");

        }

    }

}

**ExceptionHandling with MethodOverriding in Java**

#### If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception.

1. **import** java.io.\*;
2. **class** Parent{
3. **void** msg(){System.out.println("parent");}
4. }
6. **class** TestExceptionChild **extends** Parent{
7. **void** msg()**throws** IOException{
8. System.out.println("TestExceptionChild");
9. }
10. **public** **static** **void** main(String args[]){
11. Parent p=**new** TestExceptionChild();
12. p.msg();
13. }
14. }

**Output:**

**Error**

#### If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but can declare unchecked exception.

1. **import** java.io.\*;
2. **class** Parent{
3. **void** msg(){System.out.println("parent");}
4. }
6. **class** TestExceptionChild1 **extends** Parent{
7. **void** msg()**throws** ArithmeticException{
8. System.out.println("child");
9. }
10. **public** **static** **void** main(String args[]){
11. Parent p=**new** TestExceptionChild1();
12. p.msg();
13. }
14. }

#### Rule: If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

1. **import** java.io.\*;
2. **class** Parent{
3. **void** msg()**throws** ArithmeticException{System.out.println("parent");}
4. }
6. **class** TestExceptionChild2 **extends** Parent{
7. **void** msg()**throws** Exception{System.out.println("child");}
9. **public** **static** **void** main(String args[]){
10. Parent p=**new** TestExceptionChild2();
11. **try**{
12. p.msg();
13. }**catch**(Exception e){}
14. }
15. }

Output:

Cannot compile

**subclass overridden method declares same exception**

1. **import** java.io.\*;
2. **class** Parent{
3. **void** msg()**throws** Exception{System.out.println("parent");}
4. }
6. **class** TestExceptionChild3 **extends** Parent{
7. **void** msg()**throws** Exception{System.out.println("child");}
9. **public** **static** **void** main(String args[]){
10. Parent p=**new** TestExceptionChild3();
11. **try**{
12. p.msg();
13. }**catch**(Exception e){}
14. }
15. }

Output:

Child

**subclass overridden method declares subclass exception**

1. **import** java.io.\*;
2. **class** Parent{
3. **void** msg()**throws** Exception{System.out.println("parent");}
4. }
6. **class** TestExceptionChild4 **extends** Parent{
7. **void** msg()**throws** ArithmeticException{System.out.println("child");}
9. **public** **static** **void** main(String args[]){
10. Parent p=**new** TestExceptionChild4();
11. **try**{
12. p.msg();
13. }**catch**(Exception e){}
14. }
15. }

Output:

child

1. **import** java.io.\*;
2. **class** Parent{
3. **void** msg()**throws** Exception{System.out.println("parent");}
4. }
6. **class** TestExceptionChild5 **extends** Parent{
7. **void** msg(){System.out.println("child");}
9. **public** **static** **void** main(String args[]){
10. Parent p=**new** TestExceptionChild5();
11. **try**{
12. p.msg();
13. }**catch**(Exception e){}
14. }
15. }

Output:

Child

# Java Custom Exception

1. **class** InvalidAgeException **extends** Exception{
2. InvalidAgeException(String s){
3. **super**(s);
4. }
5. }
6. **class** TestCustomException1{
8. **static** **void** validate(**int** age)**throws** InvalidAgeException{
9. **if**(age<18)
10. **throw** **new** InvalidAgeException("not valid");
11. **else**
12. System.out.println("welcome to vote");
13. }
15. **public** **static** **void** main(String args[]){
16. **try**{
17. validate(13);
18. }**catch**(Exception m){System.out.println("Exception occured: "+m);}
20. System.out.println("rest of the code...");
21. }
22. }