✡ YouTube → Faisal Memon

Exception Handling → The exception handling in Java is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained

✡ **What is exception** → In Java. exception is an event the disrupts the normal flow of the Program. It is an Object which is thrown at runtime.

⊙→ **Advantage of Exception Handling:**

The core advantage of exception handling is to maintain the normal flow of the application.

✡ Type of Exception

There are mainly two type of exception; Checked and unchecked where 'error" is Considered as unchecked exception.

- The Sun microsystem says there are three type of exceptions.

(1) CheckedException
(2) UncheckedException
(3) error.

✱ Difference between checked and unchecked exceptions

(1) Checked Exception: The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions
E.g → SQLException, IOException,
Checked exceptions are checked at compile-tim

(2) → Unchecked Exception: The classes that extend RuntimeException are known as Unchecked exceptions. e.g → ArithmeticExceptio NullPointerException, ArrayIndexOutOfBoundsExceptio etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime

(3) Error : → Error is irrecoverable

e.g → OutOfMemoryError. VirtualMachineError. AssertionError etc.

Checked and UnChecked Exception

Checked Exceptions	Unchecked Exceptions
• Exception which are checked at Compile time called Checked Exception. If a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword. Examples :-) IOException) SQLException) DataAccessException) ClassNotFoundException) InvocationTargetException) MalformedURLException	• Exceptions whose handling is NOT verified during compile time. • These exceptions are handled at run time → e.g → by JVM after they occurred by using the try and catch block. • Examples: • NullPointerException • ArrayIndexOutOfBound • IllegalArgumentException • IllegalStateException

✱ Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

- "Java try block must be followed by either catch or finally block."

⊙ Syntax of java try-catch

```
try {
  // code that may throwexception
} catch (Exception_class_Name ref) { }
```

⊙ Syntax of try-finally block

```
try {
  // code that may throwexception
} finally { }
```

✱ Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only. You can use multiple catch block with a single try.

Problem without exception handling

Let's try to understand the problem if we do not use try-catch block.

Eg.→
```
Public class Testtrycatch1{
  Public static void main(String args[]){
    int data = 50/0; //may throw exception
  System.out.println("rest of the code...");
  }}
```

Output: Exception in thread main Java.lang.ArithmeticException:/byzero

As displayed in the above example, rest of the code is not executed.

"There can be 100 lines of code after exception. So all the code after exception will not be executed."

* Solution by exception handling:

Let's see the solution of above problem by Java try-catch block.

E.g-2:

```
Public class Testtrycatch2 {
public static void main (String [] args) {
try {
    int data = 50/0;
}catch (ArithmeticException e){
    System.out.println (e); }
    System.out.println("rest of the code...")
}}
```

Output: Exception in thread main: java.lang.
ArithmeticException :/ by zero.
rest of the code...

" Now, as displayed :"

✡ Java Multi catch block:

If you have to perform different tasks at the occurrence of different Exceptions, Use java multi-catch block.

e.g:

```
public class TestMultipleCatchBlock {
    Public static void main (String [] args) {
try {
        int a [] = new int [5];
        a[5] = 30/0;
    }
Catch (ArithmeticException e) {
    System. out. println ("task1 is completed");
        }
Catch ( ArrayIndexOutOfBoundsException e)
{ System. out. println ("task2 complete
}
Catch(Exception e) {
    System. out. println ("common task ");
}
System. out. println ("rest of the code");
}}
```

Output:

```
Task1 completed
rest of the code...
```

Java nested try example:

Let's see a simple example of Java nested try block.

eg→

```
class Excep6{
  Public static void main(String[] args){
  try{ try{
       System.out.println("going to divide");
         int b = 39/0;
    } catch(ArithmeticException e){
      System.out.println(e);
        }
  try{
       int a[] = new int[5];
       a[5] = 4;
} catch(ArrayIndexOutBoundException e){
  System.out.println(e); }
  System.out.println("Other Statement");
} catch(Exception e){
   System.out.println("handeled");
     }
   System.out.println("normal flow");
}
}
```

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.

Usage of Java finally

Let's see the Java finally example where exception does not occur.

```
class TestFinallyBlock {
    Public static void main(String[]args){
try{
    int data=25/5
    System.out.println(data);
}
catch (NullPointerException e){
    System.out.println(e);}
Finally{
    System.out.println("finally block is
```

```
always executed");
    }
System.out.println("rest of the code...");
    }
}
```

Output: "finally block is always executed
rest of the code..."

Java throw keyword:

The Java throw keyword is used to explicitly throw an exception.

"We can throw either checked or Uncheked exception in Java by throw keyword.

The throw keyword is mainly used to throw custom exception.

E.g:

```
Public class TestThrow1 {
    Static void validate(int age) {
        if (age<18)
    throw new ArithmeticException("notvalid
    else
    System.out.println("welcome to vote");
}
```

```
Public static void main(String []args){
    Validate(13);
System.out.println("rest of the code...");
  }
}
```

Output: Exception in thread main java.lang ArithmeticException: not valid.

Java throws keyword:

The Java throws keyword is used to declare an exception. It gives an information to the Programmer that there may occur an exception so it is better for the Programmer to provide the exception handling code so that normal flow can be maintained.

* Syntax of Java throws

```
return_type method_name ( ) throw exception_
    class_name{
    //method
    }
```

Java throws Example

```
import java.io.IOException;
class Testthrows1 {
  Void m () throws IOException {
      throw new IOException("device error");
        // checked exception
    }
Void n () throws IOException {
        m();
    }
Void p(){
  try {
      n();
  }catch(Exception e){
   System.out.println("exception handled");
        }
    }
public static void main(String []args){
   Testthrows1 obj = new Testthrows1();
   obj.p();
   System.out.println("normal flow...");
    }
}
```

Output:

```
exception handled
normal flow ...
```

✡ Java Custom Exception

If you are creating your own Exception that is known as custom exception or User-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and massage.

Example – "In screenshot → github repo"

" If you want to take the Class-based example then follow → the GitHub repository "

* Java - Exception Interview Question:

(1) What is the difference between Checked and Unchecked

(2) What is the difference between throw and throws?

(3) What happen if an exception is not handled in Java?

(4) Can we have multiple catch blocks in Java?

(5) What is the difference between final, finally and finalize()?

Figure: Exception Hierarchy in Java