Kotlin Exception Handling:

Exception is a runtime problem which occurs in the program and leads to program termination.

This may be occure due to *running out of memory space*, *array out of bond*, *condition like divided by zero*.

To handle this type of problem during program execution the technique of exception handling is used.

Exception handling is a technique which handles the runtime problems and maintains the flow of program execution.

In Kotlin, all exception classes are descendants of class **Throwable**.
To throw an exception object, Kotlin uses the ***throw*** expression.

```
throw MyException("this throws an exception")
```

There are **four** different keywords used in exception handling. These are:

- try
- catch
- finally
- throw

try: try block contains set of statements which might generate an exception. It must be followed by either catch or finally or both.

catch: catch block is used to catch the exception thrown from try block.

finally: finally block always execute whether exception is handled or not. So it is used to execute important code statement.

throw: throw keyword is used to *throw an exception explicitly*.

Kotlin Unchecked Exception

Unchecked exception is that exception which is thrown due to mistakes in our code.

This exception type extends ***RuntimeException*** class.

The Unchecked exception is checked at run time.

Following are some example of unchecked exception:

- **ArithmeticException**: thrown when we divide a number by zero.
- **ArrayIndexOutOfBoundExceptions**: thrown when an array has been tried to access with incorrect index value.
- **SecurityException**: thrown by the security manager to indicate a security violation.
- **NullPointerException**: thrown when invoking a method or property on a null object.

Checked Exception in Java

Checked exception is checked at compile time.

This exception type extends the **Throwable** class.

Following are some example of unchecked exception:

- IOException.
- SQLException etc.

Note: Kotlin does not support checked exception.

Kotlin try catch

Kotlin try-catch block is used for exception handling in the code.

The **try** block encloses the code which may **throw** an exception and the catch block is used to handle the exception.

This block must be written within the method.

Kotlin **try** block must be followed by either catch block or finally block or both.

Syntax of try with catch block

```
try{
    //code that may throw exception
    }catch(e: SomeException){
    //code that handles exception
    }
```

Syntax of try with finally block

```
try{
    //code that may throw exception
    }finally{
    // code finally block
    }
```

Syntax of try catch with finally block

```
try{
    // some code
    }
    catch (e: SomeException) {
        // handler
    }
    finally {
        // optional finally block
    }
```

Problem without Exception Handling

Lets's see an example which causes exception which is not handled.

```
fun main(args: Array<String>){
    val data = 20 / 0   //may throw exception
    println("code below exception ...")
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionHandlingKt.main(ExceptionHandling.kt:2)
```

Solution by exception handling

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

```
fun main(args: Array<String>){
    try {
        val data = 20 / 0  //may throw exception
    } catch (e: ArithmeticException) {
        println(e)
    }
    println("code below exception...")
}
```

Output:

```
java.lang.ArithmeticException: / by zero
code below exception...
```

Kotlin try block as an Expression

We can use **try** block as an *expression* which returns a value.

The value returned by **try** *expression* is either the last *expression* of **try** block or the last *expression* of catch.

Contents of the *finally* block do not affect the result of the expression.

Kotlin try as an expression example

Let's see an example of **try-catch** block **as** an expression which returns a value.

In this example **String** value to Int which does not generate any exception and returns last statement of try block.

```
fun main(args: Array<String>){
     val str = getNumber("10")
     println(str)
}
fun getNumber(str: String): Int{
    return try {
        Integer.parseInt(str)
    } catch (e: ArithmeticException) {
        0
    }
}
```

Let's modify the above code which generate an exception and return the last statement of catch block.

```
fun main(args: Array<String>){
     val str = getNumber("10.5")
     println(str)
}
fun getNumber(str: String): Int{
    return try {
        Integer.parseInt(str)
    } catch (e: NumberFormatException) {
        0
    }
}
```

Kotlin Multiple catch Block

We can use multiple catch block in our code.
Kotlin multiple catch blocks are used when we are using different types of operation in try block which may causes different exceptions in try block.

Kotlin multiple catch block example 1

Let's see an example of multiple catch blocks.
In this example we will are performing different types of operation.
These different types of operation may generate different types of exceptions.

```
fun main(args: Array<String>){
    try {
        val a = IntArray(5)
        a[5] = 10 / 0
    }catch (e: ArithmeticException) {
        println("arithmetic exception catch")
    }catch (e: ArrayIndexOutOfBoundsException) {
        println("array index outofbounds exception")
    }catch (e: Exception) {
        println("parent exception class")
    }
    println("code after try catch...")
}
```

Output:

```
arithmetic exception catch
code after try catch...
```

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

Kotlin Nested try-catch block

- We can also able to use nested try block whenever required.
- Nested try catch block is such block in which one try catch block is implemented into another try block.
- The requirement of nested try catch block is arises when a block of code generates an exception and within that block another code statements also generates another exception.

Syntax of nested try block

```
..
try
{
    // code block
    try
    {
        // code block
    }
    catch(e: SomeException)
    {
    }
}
catch(e: SomeException)
{
}
..
```

Kotlin nested try block example

```
fun main(args: Array<String>) {
    val nume = intArrayOf(4, 8, 16, 32, 64, 128, 256, 512)
    val deno = intArrayOf(2, 0, 4, 4, 0, 8)
    try {
        for (i in nume.indices) {
            try {
                println(nume[i].toString() + " / " +
                deno[i] + " is " + nume[i] / deno[i])
            }catch (exc: ArithmeticException) {
                println("Can't divided by Zero!")
            }

        } // For loop over
    }catch (exc: ArrayIndexOutOfBoundsException) {
        println("Element not found.")
    }
}
```

Output:

```
4 / 2 is 2
Can't divided by Zero!
16 / 4 is 4
32 / 4 is 8
Can't divided by Zero!
128 / 8 is 16
Element not found.
```

Kotlin finally Block

Kotlin finally block such block which is always executes whether exception is handled or not.
So it is used to execute important code statement.

Kotlin finally Block Example 1

Let's see an example of exception handling in which exception does not occur.

```
fun main (args: Array<String>){
    try {
        val data = 10 / 5
        println(data)
    } catch (e: NullPointerException) {
        println(e)
    } finally {
        println("finally block always executes")
    }
    println("below codes...")
}
```

Output:

```
2
finally block always executes
below codes...
```

Kotlin finally Block Example 2

Let's see an example of exception handling in which exception occurs but not handled.

```
fun main (args: Array<String>){
    try {
        val data = 5 / 0
        println(data)
    } catch (e: NullPointerException) {
        println(e)
    } finally {
        println("finally block always executes")
    }
    println("below codes...")
}
```

Output:

```
finally block always executes
Exception in thread "main" java.lang.ArithmeticException: / by zero
```

Kotlin finally Block Example 3

Let's see an example of exception handling in which exception occurs and handled.

```
fun main (args: Array<String>){
    try {
        val data = 5 / 0
        println(data)
    } catch (e: ArithmeticException) {
        println(e)
    } finally {
        println("finally block always executes")
    }
    println("below codes...")
}
```

Output:

```
java.lang.ArithmeticException: / by zero
finally block always executes
below codes
```

Note: The finally block will not be executed if program exits (either by calling exitProcess(Int) or any error that causes the process to abort).

Kotlin throw keyword

Kotlin throw keyword is used to throw an explicit exception. It is used to throw a custom exception.

To throw an exception object we will use the throw-expression.

Syntax of throw keyword

```
throw SomeException()
```

Kotlin throw example

Let's see an example of ***throw*** keyword in which we are validating age limit for driving license.

```
fun main(args: Array<String>) {
    validate(15)
    println("code after validation check...")
}
fun validate(age: Int) {
    if (age < 18)
        throw ArithmeticException("under age")
    else
        println("eligible for drive")
}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: under age
```