EXCEPTIONAL HANDLING IN JAVA

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.

Advantage of Exception Handling

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. Let's take a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Advantage of Exception Handling

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. Let's take a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

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.

TRY CATCH example:

```
public class JavaExceptionExample{
  public static void main(String args[]){
   try{
     //code that may raise exception
     int data=100/0;
   }catch(ArithmeticException e){System.out.println(e);}
   //rest code of the program
   System.out.println("rest of the code...");
  }
}
```

Output:

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

Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

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, 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 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

FINALLY example:

The finally statement lets you execute code, after try...catch, regardless of the result:

Example

```
public class MyClass {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
```

```
    System.out.println("The 'try catch' is finished.");
  }
 }
}
```

The output will be:

Something went wrong.
The 'try catch' is finished.

THROW example:

The throw statement allows you to create a custom error.

The throw statement is used together with an **exception type**. There are many exception types available in Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, etc:

Example

Throw an exception if **age** is below 18 (print "Access denied"). If age is 18 or older, print "Access granted":

```
public class MyClass {
  static void checkAge(int age) {
    if (age < 18) {
```

```
    throw new ArithmeticException("Access denied - You must be at least 18 years old.");
  }
  else {
    System.out.println("Access granted - You are old enough!");
  }
 }


 public static void main(String[] args) {
  checkAge(15); // Set age to 15 (which is below 18...)
 }
}
```

The output will be:

```
Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least 18 years old.
        at MyClass.checkAge(MyClass.java:4)
        at MyClass.main(MyClass.java:12)
```

If **age** was 20, you would **not** get an exception:

```
checkAge(20);
```

The output will be:

Access granted - You are old enough!

THROWS

throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.
Syntax:
type method_name(parameters) throws exception_list
exception_list is a comma separated list of all the
exceptions which a method might throw.