­­­LECTURE 25

TYPES OF EXCEPTION IN JAVA

1. **Checked Exceptions:**

* Checked exceptions are exceptions that are checked at compile-time.
* Any method that might throw a checked exception must declare it using the throws keyword in its method signature.
* Checked exceptions are typically used for conditions that a program can reasonably be expected to recover from.
* Examples of checked exceptions include:
  + IOException: Signals that an I/O exception of some sort has occurred.
  + SQLException: Indicates that an error occurred while accessing a database.
  + ClassNotFoundException: Indicates that a class could not be found during runtime.

1. **Unchecked exceptions,**

* Unchecked exceptions also known as runtime exceptions, are not checked at compile-time.
* Methods are not required to declare unchecked exceptions in their throws clause.
* Unchecked exceptions usually indicate programming errors or unexpected conditions that are beyond the programmer's control.
* Examples of unchecked exceptions include:
* NullPointerException: Thrown when attempting to access an object reference that is null.
* ArrayIndexOutOfBoundsException: Thrown when trying to access an array element with an invalid index.
* ArithmeticException: Thrown when an arithmetic operation encounters an error, such as division by zero.

Java Exception Handling Example

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

**JavaExceptionExample.java (Unchecked)**

**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...");

  }

}

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](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

If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a [string](https://www.javatpoint.com/java-string) variable that has characters; converting this variable into digit will cause NumberFormatException.

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

4) A scenario where ArrayIndexOutOfBoundsException occurs

When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException. Consider the following statements.

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

Example:

**public** **class** Main {

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

**try** {

// code that may raise exception

**int**[] numbers = {1, 2, 3};

System.***out***.println(numbers[3]);

} **catch** (ArrayIndexOutOfBoundsException e) {

System.***out***.println(e);

}

// rest code of the program

System.***out***.println("rest of the code...");

}

}

EXCEPTION HANDLING IN JAVA

Java Exceptions - Try...Catch

Java Exceptions

When executing Java code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable things.

When an error occurs, Java will normally stop and generate an error message. The technical term for this is: Java will throw an **exception** (throw an error).

Java try and catch

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs:

Syntax

try {

// *Block of code to try*

}

catch(Exception *e*) {

// *Block of code to handle errors*

}

Consider the following example:

This will generate an error, because **myNumbers[10]** does not exist.

public class Main {

public static void main(String[ ] args) {

int[] myNumbers = {1, 2, 3};

System.out.println(myNumbers[10]); // error!

}

}

The output will be something like this:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10  
        at Main.main(Main.java:4)

**Note:** ArrayIndexOutOfBoundsException occurs when you try to access an index number that does not exist.

If an error occurs, we can use try...catch to catch the error and execute some code to handle it:

Example

public class Main {

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.");

}

}

}

The output will be:

Something went wrong.

Finally

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

Example

public class Main {

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.

The throw keyword

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 Main {

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 Main.checkAge(Main.java:4)  
        at Main.main(Main.java:12)

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

Example

checkAge(20);

The output will be:

Access granted - You are old enough!

Problem without exception handling

Let's try to understand the problem if we don't use a try-catch block.

Example 1

**TryCatchExample1.java**

**public** **class** TryCatchExample1 {

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

**int** data=50/0; //may throw exception

        System.out.println("rest of the code");

    }

}

Exception in thread "main" java.lang.ArithmeticException: / by zero

As displayed in the above example, the **rest of the code** is not executed (in such case, the **rest of the code** statement is not printed).

There might be 100 lines of code after the exception. If the exception is not handled, all the code below the exception won't be executed.

Solution by exception handling

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

Example 2

**TryCatchExample2.java**

**public** **class** TryCatchExample2 {

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

**try**

        {

**int** data=50/0; //may throw exception

        }

            //handling the exception

**catch**(ArithmeticException e)

        {

            System.out.println(e);

        }

        System.out.println("rest of the code");

    }

}

**Output:**

java.lang.ArithmeticException: / by zero

rest of the code

As displayed in the above example, the **rest of the code** is executed, i.e., the **rest of the code** statement is printed.

Example 3

In this example, we also kept the code in a try block that will not throw an exception.

**TryCatchExample3.java**

**public** **class** TryCatchExample3 {

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

**try**

        {

**int** data=50/0; //may throw exception

                         // if exception occurs, the remaining statement will not exceute

        System.out.println("rest of the code");

        }

             // handling the exception

**catch**(ArithmeticException e)

        {

            System.out.println(e);

        }

    }

}

**Output:**

java.lang.ArithmeticException: / by zero

Here, we can see that if an exception occurs in the try block, the rest of the block code will not execute.

Example 5

Let's see an example to print a custom message on exception.

**TryCatchExample5.java**

**public** **class** TryCatchExample5 {

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

**try**

        {

**int** data=50/0; //may throw exception

        }

             // handling the exception

**catch**(Exception e)

        {

                  // displaying the custom message

            System.out.println("Can't divided by zero");

        }

    }

}

Example 8

In this example, we handle the generated exception (Arithmetic Exception) with a different type of exception class (ArrayIndexOutOfBoundsException).

**TryCatchExample8.java**

**public** **class** TryCatchExample8 {

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

**try**

        {

**int** data=50/0; //may throw exception

        }

            // try to handle the ArithmeticException using ArrayIndexOutOfBoundsException

**catch**(ArrayIndexOutOfBoundsException e)

        {

            System.out.println(e);

        }

        System.out.println("rest of the code");

    }

}

**Output:**

Exception in thread "main" java.lang.ArithmeticException: / by zero

Java Catch Multiple Exceptions

Java Multi-catch block

A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Points to remember

* At a time only one exception occurs and at a time only one catch block is executed.
* All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Example 1

Let's see a simple example of java multi-catch block.

**MultipleCatchBlock1.java**

**public** **class** MultipleCatchBlock1 {

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

**try**{

**int** a[]=**new** **int**[5];

                a[5]=30/0;

               }

**catch**(ArithmeticException e)

                  {

                   System.out.println("Arithmetic Exception occurs");

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                   System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

Example 2

**MultipleCatchBlock2.java**

**public** **class** MultipleCatchBlock2 {

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

**try**{

**int** a[]=**new** **int**[5];

                System.out.println(a[10]);

               }

**catch**(ArithmeticException e)

                  {

                   System.out.println("Arithmetic Exception occurs");

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                   System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

**Output:**

ArrayIndexOutOfBounds Exception occurs

rest of the code

In this example, try block contains two exceptions. But at a time only one exception occurs and its corresponding catch block is executed.