Unit 5:

Handling Error/Exceptions

Exceptions:

- Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions.
- Exceptions can be caught and handled by the program.
- When an exception occurs within a method, it creates an object. This object is called the exception object. It contains information about the exception, such as the name and description of the exception and the state of the program when the exception.

Major reasons why an exception Occurs :

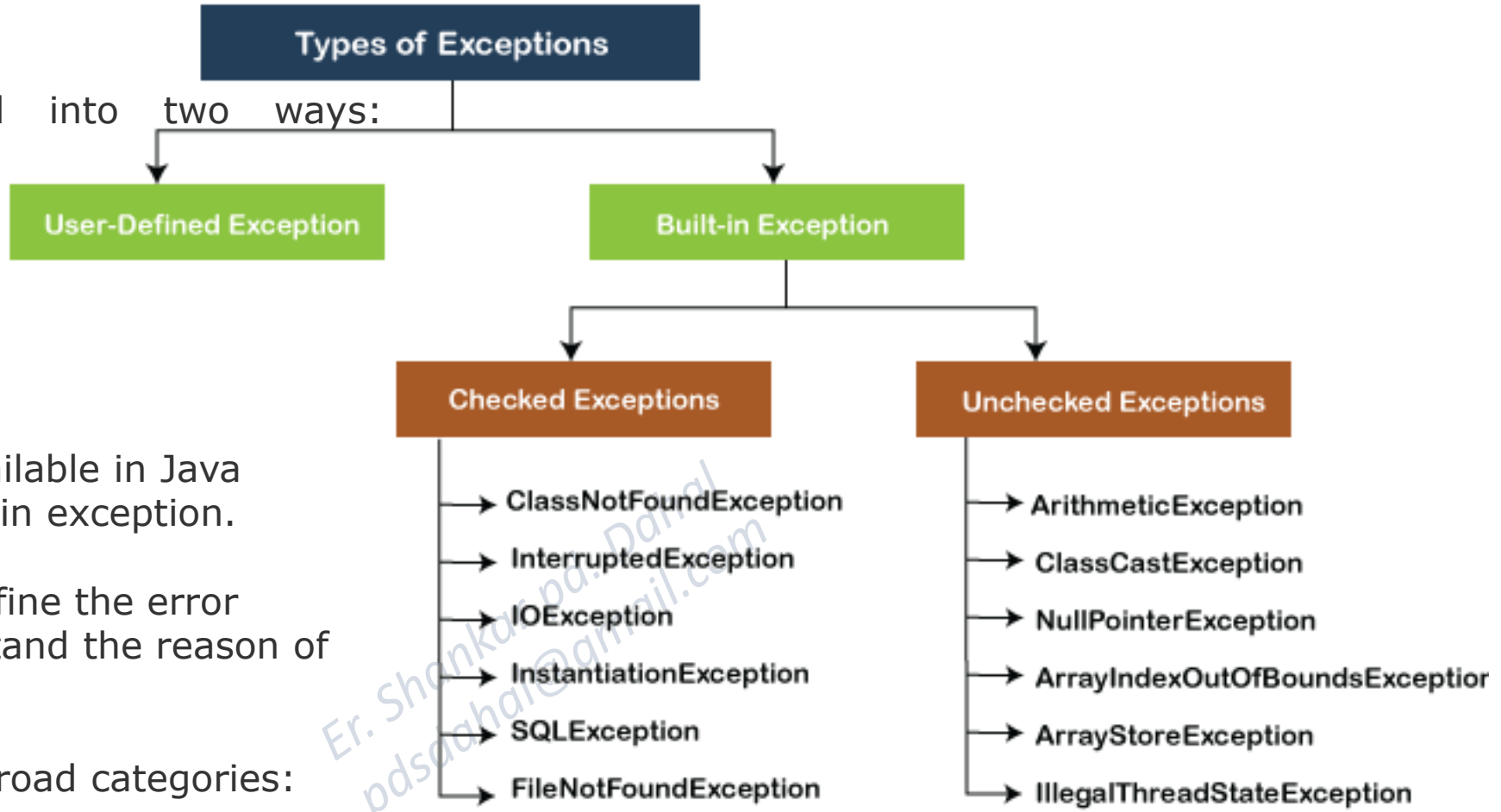
- Invalid user input
- Device failure
- Loss of network connection
- Physical limitations (out of disk memory)
- Code errors
- Opening an unavailable file

Exceptions Types in Java

Exceptions can be categorized into two ways:

1.Built-in Exceptions

2. User-Defined Exceptions



1. Built-in Exceptions

- Exceptions that are already available in Java libraries are referred to as built-in exception.
- These exceptions are able to define the error situation so that we can understand the reason of getting this error.
- It can be categorized into two broad categories:

  a. Checked Exception
  b. Unchecked Exception

1. Built-in Exceptions :

a. Checked Exception :

- Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.

b. Unchecked Exception :

- The unchecked exceptions are just opposite to the checked exceptions.
- The compiler will not check these exceptions at compile time, but they are checked at runtime.
- In simple words, if a program throws an unchecked exception, and even if we didn't handle or declare it, the program would not give a compilation error.

2. User-Defined Exceptions:

- Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called user-defined Exceptions.

Basic Scenarios of Exceptions:

1. A scenario where ArithmeticException occurs

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

3. A scenario where NumberFormatException occurs

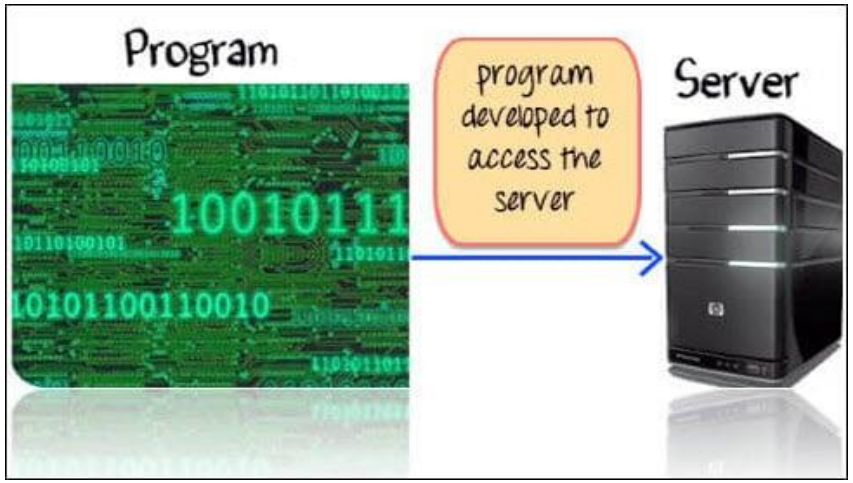
- If the formatting of any variable or number is mismatched, it may result into NumberFormatException. Suppose we have a string variable that has characters; converting this variable into digit will cause NumberFormatException.

4. A scenario where ArrayIndexOutOfBoundsException occurs

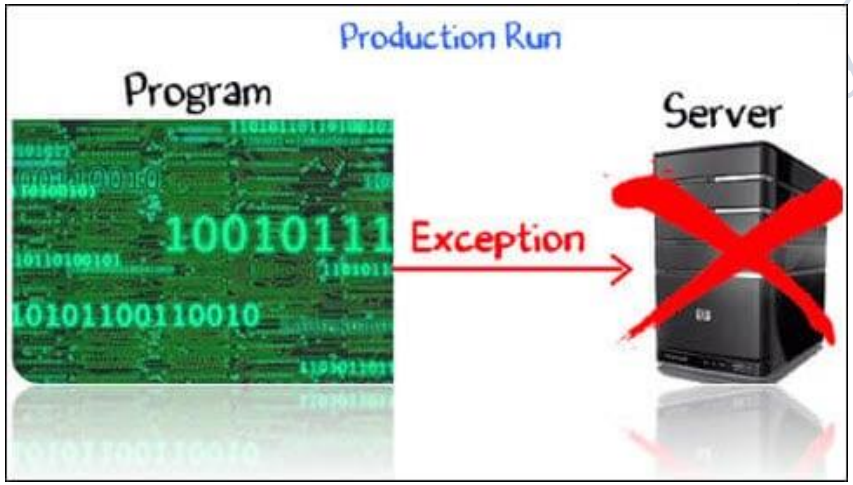
- When an array exceeds to it's size, the ArrayIndexOutOfBoundsException occurs. there may be other reasons to occur ArrayIndexOutOfBoundsException.

Why do we need Exception?

- Suppose you have coded a program to access the server. Things worked fine while you were developing the code.



- During the actual production run, the server is down. When your program tried to access it, an exception is raised.



Exception Handling in Java :

- The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that the normal flow of the application can be maintained.

- Java exception handling is managed via five keywords: try, catch, throw, throws and finally.

1. try :

- The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.

2. 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.

3. finally :

- The "finally" block is used to execute the necessary code of the program.
- It is executed whether an exception is handled or not.
- Putting cleanup code in a finally block is always a good practice, even when no exceptions are anticipated.

4. throw :

- The "throw" keyword is used to throw an exception.

5. throws :

- The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

Java try-catch block :

- The try...catch block in Java is used to handle exceptions and prevents the abnormal termination of the program.
- The try block includes the code that might generate an exception.
- The catch block includes the code that is executed when there occurs an exception inside the try block.

Syntax

```
try {
        // Block of code to try
} catch(Exception e) {
        // Block of code to handle errors
}
```


Java try...finally block

- We can also use the try block along with a finally block.
- finally block is executed whether an exception is handled or not.

Syntax :

```
try {
}finally {
}
```

Java try – catch - finally block :

Syntax:

```
try {
        //statements that may cause an exception
}catch (Exception e) {
        //statements that will execute if exception occurs
}finally {
        //statements that execute whether the exception occurs or not
}
```


Why use Java finally block?

- finally block in Java can be used to put "**cleanup**" code such as closing a file, closing connection, etc.
- The important statements to be printed can be placed in the finally block.

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.

Note :

- Starting from Java 7.0, it is possible for a single catch block to catch multiple exceptions by separating each with | (pipe symbol) in the catch block.
- Catching multiple exceptions in a single catch block reduces code duplication and increases efficiency. The bytecode generated while compiling this program will be smaller than the program having multiple catch blocks as there is no code redundancy.

Syntax :

```
try {
        //block of code

}catch(ExceptionType1 | ExceptionType2    | ExceptionType3 ex) {

        //block of code
}
```

Java Nested try block :

❖ When a try catch block is present in another try block then it is called the nested try catch block.

Syntax:

```
....
//Main try block
try {
   statement 1;
   statement 2;
   //try-catch block inside another try block
   try {
      statement 3;
      statement 4;
      //try-catch block inside nested try block
      try {
         statement 5;
         statement 6;
      }
      catch(Exception e2) {
         //Exception Message
      }
   }
   catch(Exception e1) {
       //Exception Message
   }

}
//Catch of Main(parent) try block
catch(Exception e3) {
      //Exception Message
}
....
```

Java throw :

- throw keyword is used to create an exception object manually.
- When we use throw, programmer is responsible to create an exception object.
- In case of throw keyword, we can throw only single exception.
- throw keyword is used with the method.

Syntax:

```
throw new exception_class("error message");
```

Java throws :

- throws is a keyword used in the method signature used to declare an exception which might get thrown by the function while executing the code.

- throws keyword is followed by exception class names.

- It indicates the caller method that given exception can occur, so we have to handle it either using try catch block or again declare it by using throws keyword.

- In case of throws keyword, we can declare multiple exceptions.

Syntax:

```
return_type method_name() throws exception_class_name_1, exception_class_name_2, ... {

}
```

Difference between Errors and Exceptions in Java

S.No	Errors	Exceptions
1.	The error indicates trouble that primarily occurs due to the scarcity of system resources.	The exceptions are the issues that can appear at runtime and compile time.
2.	It is not possible to recover from an error.	It is possible to recover from an exception.
3.	In java, all the errors are unchecked.	In java, the exceptions can be both checked and unchecked.
4.	The system in which the program is running is responsible for errors.	The code of the program is accountable for exceptions.
5.	They are described in the java.lang.Error package.	They are described in java.lang.Exception package