EXCEPTIONAL HANDLING IN JAVA

WHAT IS EXCEPTION IN JAVA?

An exception is an unwanted or unexpected event that occurred during the execution of the program, that disrupts the flow of the program. For example, if a user is trying to divide an integer by 0 then it is an exception, as it is not possible mathematically.

WHY WE HANDLE JAVA EXCEPTION?

We handle exceptions in java to make sure the program executes properly without any halt, which occurs when an exception is raised. These exceptions are runtime as well as compile-time, they may be minor exceptions but can cause a problem in executions of the program, hence it becomes necessary to handle java exceptions. If java exceptions are not handled, programs may crash and execution is halted in between.

For example, if there is a program with some lines of code and an exception occurs mid-way after executing certain lines of code, in this case, the execution terminates abruptly. But if the programmer handles the exceptions explicitly, all the statements of code are executed properly and the flow of the program is maintained.

Let's see a code for the same.

```
class SampleCode
{
	public static void main (String args[])
	{
		System.out.println("Hello World!");
	  int a = 10;
		int b = 0;
		System.out.println(a/b);
	  System.out.println("Welcome to java programming.");
	  System.out.println("Bye.")
	}
}
```

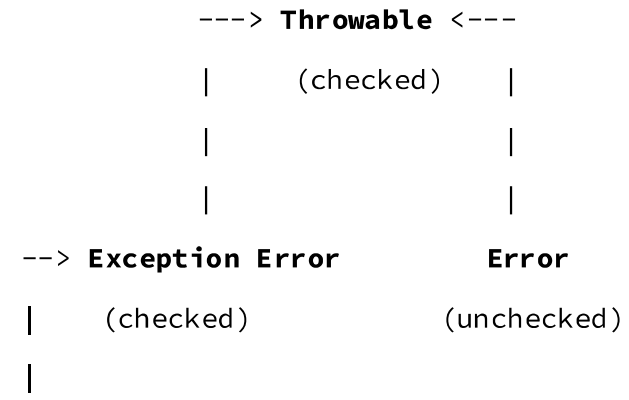
OUTPUT:

```
Hello World!
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at SampleException.main(SampleException.java:8)
```

In the above code, the first three lines in the main method are executed properly. At the 4th line, an integer is divided by 0, which is not possible and an exception is raised by JVM (Java Virtual Machine). In this case, the exception is not handled by the programmer which will halt the program in between by throwing the exception, and rest of the lines of code won't be executed.

Hierarchy of Java Exception Classes

As java is an object-oriented language every class extends the **Object class**. All exceptions and errors are subclasses of class Throwable. Throwable is the base/superclass of exception handling hierarchy in java. This class is further extended into different classes like exception, error and so on.



Types of Exceptions Handling in Java

1. **Checked Exception**

Checked exceptions are checked at compile time. If a method throws a checked exception, then the throws clause of the method signature must specify the exception. And the code will not compile without this.

On the caller side, the method that calls a method that can throw a checked exception must either,

- Handle the exception (with a catch block)
- Declare the checked exception in its throws clause (propagating it back to its caller).

2. **Unchecked Exception**

Unchecked exceptions are not checked during compile time. Hence, the method of throwing an unchecked exception will not (need not) declare it in its signature. Few examples unchecked exceptions in Java:

- ArithmeticException
- NullPointerException
- IndexOutOfBoundException

Let's understand the meaning of the term's exception and error.

ERROR

Error is the wrongs that can make a program go wrong. An error may produce an incorrect output or may terminate the execution of the program or even may cause the system to crash.

TYPES OF ERROR

1. **Compile-time error**

All syntax errors will be detected and displayed by the java compiler. Whenever the compiler displays an error, it will not create the .class file. So, it is necessary that we fix all the errors before we can successfully compile and run the program. Most of the compile time errors are due to spelling mistakes. Basically, compile errors are the errors generated at the time of program compilation.

2. **Run-time error**

Sometimes, a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic. Most common run time errors are:

- Accessing an element that is out of bounds of an array
- Accessing a character that is out of bounds of a string
- Trying to store a value into the array of an incompatible class or type
- Attempting to use a negative size for an array

EXCEPTION

An Exception is a condition that is caused by a run time error in the program. When java interpreter encounters an error, such as dividing an integer by zero, it creates an exception object and throws it then takes the correct action. This task is known as **exception handling.**

This mechanism performs the following tasks:

1. Find the problem (Hit the exception).
2. Inform them that an error has occurred (Throw the exception).
3. Receive the error information (Catch the exception).
4. Take corrective actions (Handles the exception).

The error handling code consists of two segments, one to detect errors and to throw exceptions and the other to catch exceptions and to take appropriate actions.

TYPES OF EXCEPTION

Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.

1. **ArithmeticException**: It is thrown when an exceptional condition has occurred in an arithmetic operation.

2. **ArrayIndexOutOfBoundsException**: It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

3. **ClassNotFoundException**: This Exception is raised when we try to access a class whose definition is not found

4. **FileNotFoundException**: This Exception is raised when a file is not accessible or does not open.

5. **IOException**: It is thrown when an input-output operation fails or interrupts.

6. **InterruptedException**: It is thrown when a thread is waiting, sleeping, or doing some processing, and it is interrupted.

7. **NoSuchFieldException**: It is thrown when a class does not contain the field (or variable) specified

8. **NoSuchMethodException**: It is thrown when accessing a method which is not found.

9. **NullPointerException**: This exception is raised when referring to the members of a null object. Null represents nothing

10. **NumberFormatException**: This exception is raised when a method could not convert a string into a numeric format.

11. **RuntimeException**: This represents any exception which occurs during runtime.

12. **StringIndexOutOfBoundsException**: It is thrown by String class methods to indicate that an index is either negative than the size of the string

Handling Exceptions

In java customized exception handling is achieved using five keywords: try, catch, throw, throws, and finally. Here are how these keywords work in short.

- Try block contains the program statements that may raise an exception.
- Catch block catches the raised exception and handles it.
- Throw keyword is used to explicitly throw an exception.
- Throws keyword is used to declare an exception.
- Finally block contains statements that must be executed after the try block.

Syntax of Exception Handling Code

Java uses a keyword try to write a block of code that is likely to cause an error condition and "throw" an exception. Catch block is defined by the keyword catch, it catches the exception thrown by the try block and handle it properly the catch block is added immediately after the try block.

Example:

```
.....

.....
try
{
 Statement;                                    // generates an exception
}
catch (Exception-type e)
{
Statements;                                    // processing the exception
}
.....
.....
// Program to demonstrate exception handling mechanism
// Java program to demonstrate ArithmeticException
class Demo
{
   public static void main (String args[])
   {
     try
     {
        int a = 30, b = 0; int c = a/b;              // cannot divide by zero
        System.out.println ("Result = " + c);
     }
     catch(ArithmeticException e)
     {
        System.out.println ("Can't divide a number by 0");
```

```
    }
  }
}
```

Output:

Can't divide a number by 0

Java Exception Handling Best Practices

- Always catch only those exceptions that can be handled.
- Null values should not be returned in catch block rather than handling an exception, it consumes the exception and the error fails permanently and programs crashes.
- Don't get the exception class instead catch particular subclasses.
- Never throw an exception from the finally block, because even if any exception is not raised in the try block, finally block will throw the exception, after execution of try-catch blocks.
- Use finally blocks instead of catch blocks if you are not going to handle the exception.
- Always use the finally block to close the used resources.

Points need to remember

- try block should have at least one catch or one finally block for execution.
- If there is the possibility of multiple exceptions being raised multiple and respective catch blocks should be used.
- 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 sequentially.
- Most important use of finally block is to close the used data resources.
- throw keyword is mainly used to throw custom exceptions.
- Custom exceptions are used to make the program user-friendly.
- Final, finally, and finalize are three different keywords with completely different functionalities in java.