Last Updated : 20 Dec, 2024

Chained Exceptions in Java allow associating one exception with another, i.e. one exception describes the cause of another exception. For example, consider a situation in which a method throws an ArithmeticException because of an attempt to divide by zero, but the root cause of the error was an I/O failure that caused the divisor to be zero. In such cases, chained exceptions help propagate both the primary and underlying causes of the error.

Example: The below example demonstrates ***how to use chained exceptions in Java***.

```java
// Working of chained exceptions
public class Geeks {
    public static void main(String[] args) {
        try {

            // Creating an exception
            NumberFormatException ex = new NumberFormatException("Primary
Exception");

            // Setting the cause of the exception
            ex.initCause(new NullPointerException("Root cause of the
exception"));

            // Throwing the exception with a cause
            throw ex;
        }
        catch (NumberFormatException ex) {

            // Displaying the primary exception
            System.out.println("Caught Exception: " + ex);

            // Displaying the root cause of the exception
            System.out.println("Cause of Exception: " + ex.getCause());
        }
    }
}
```

Output

```
Caught Exception: java.lang.NumberFormatException: Primary
Exception
Cause of Exception: java.lang.NullPointerException: Root cause of
the exception
```

Chained exceptions, also known as nested exceptions, allow us to associate a cause with an exception in Java. This is useful when we want to propagate information about the original cause of an exception.

Constructors of Throwable Supporting Chained Exceptions

1. **Throwable(Throwable cause)**: Where cause is the exception that causes the current exception.
2. **Throwable(String msg, Throwable cause)**: Where **msg** is the exception message and cause is the exception that causes the current exception.

Methods of Throwable Supporting Chained Exceptions

1. getCause() method: This method returns actual cause of an exception.
2. initCause(Throwable cause) method: This method sets the cause for the calling exception.

Example: Using a Custom Message with Chained Exceptions

In Java, we can chain exceptions using the constructor of the Throwable class.

```
// Use a custom message with chained exception
public class Geeks {
    public static void main(String[] args) {
        try {

            // Code that might throw an exception
            int[] n = new int[5];
            int divisor = 0;
```

```
                for (int i = 0; i < n.length; i++) {
                    int res = n[i] / divisor;
                    System.out.println(res);
                }
            }
            catch (ArithmeticException e) {

                // Creating a new exception with
                // the original as the cause
                throw new RuntimeException
                  ("Error: Division by zero occurred", e);
            }
        }
    }
```

Output:

```
Hangup (SIGHUP)
Exception in thread "main" java.lang.RuntimeException: Error:
Division by zero occurred
    at Geeks.main(Geeks.java:18)
Caused by: java.lang.ArithmeticException: / by zero
    at Geeks.main(Geeks.java:10)
```

Explanation: In this example, an array of integers and sets the divisor to 0. Inside the **try** block. It try to divide each element of the array by 0, which throws an **ArithmeticException**. This ArithmeticException is caught in the **catch** block, where a new RuntimeException is created with the original exception i.e. ArithmeticException as its cause. Since the RuntimeException is not caught, which displays the stack trace, including the RuntimeException and the ArithmeticException.

Advantages of Chained Exceptions

- This exception helps in debugging by providing details about both primary and root causes.
- It simplifies error handling by enabling propagation of complete exception context.
- This improves traceability of errors in complex applications.

Kickstart your Java journey with our online course on Java Programming, covering everything from basics to advanced concepts.