Java Chained Exceptions

In this article, we will discuss how the chained exception feature allows you to associate another exception with an exception.

Java Chained Exceptions Overview

The chained exception feature allows you to associate another exception with an exception. This second exception describes the cause of the first exception.

For example, imagine a situation in which a method throws an `ArithmeticException` because of an attempt to divide by zero. However, the actual cause of the problem was that an I/O error occurred, which caused the divisor to be set improperly. Although the method must certainly throw an `ArithmeticException`, since that is the error that occurred, you might also want to let the calling code know that the underlying cause was an I/O error. Chained exceptions let you handle this, and any other situation in which layers of exceptions exist. This concept was introduced in JDK 1.4.

Throwable Class

Throwable class has some constructors and methods to support chained exceptions. Firstly, let's look at the constructors.

1. Throwable(Throwable cause) – Throwable has a single parameter, which specifies the actual cause of an Exception.
2. Throwable(String desc, Throwable cause) – this constructor accepts an Exception description with the actual cause of an Exception as well.

Next, let's have a look at the methods this class provides:

- getCause() method – This method returns the actual cause associated with the current Exception.
- initCause() method – It sets an underlying cause with invoking Exception.

Read more about Throwable Class on **java.lang.Throwable Class in Java**

Java Chained Exceptions Example

Here is an example that illustrates the mechanics of handling chained exceptions:

```
// Demonstrate exception chaining.
class ChainExcDemo {
     static void demoproc() {
         // create an exception
         NullPointerException e = new NullPointerException("top layer");
         // add a cause
         e.initCause(new ArithmeticException("cause"));
         throw e;
     }

    public static void main(String args[]) {
        try {
             demoproc();
         } catch (NullPointerException e) {
             // display top level exception
               System.out.println("Caught: " + e);
              // display cause exception
              System.out.println("Original cause: " + e.getCause());
         }
    }
}
```

Output:

```
Caught: java.lang.NullPointerException: top layer
Original cause: java.lang.ArithmeticException: cause
```

In this example, the top-level exception is *NullPointerException*. To it is added a cause exception, *`ArithmeticException`*. When the exception is thrown out of *`demoproc( )`* method, it is caught by *`main( )`*. There, the top-level exception is displayed, followed by the underlying exception, which is obtained by calling *`getCause( )`*.

Chained exceptions can be carried on to whatever depth is necessary. Thus, the cause exception can, itself, have a cause. Be aware that overly long chains of exceptions may indicate poor design.

Chained exceptions are not something that every program will need. However, in cases in which knowledge of an underlying cause is useful, they offer an elegant solution.