CHAINED EXCEPTIONS

Chained Exceptions allows **to relate one exception with another exception**, i.e one exception describes 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 actual cause of exception was an I/O error which caused the divisor to be zero. The method will throw only ArithmeticException to the caller. So the caller would not come to know about the actual cause of exception. Chained Exception is used in such type of situations.

Throwable constructors that supports chained exceptions are:

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.

Throwable methods that supports chained exceptions are:

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:

```
import java.io.IOException;
public class ChainedException
 {
  public static void divide(int a, int b)
  {
   if(b==0)
   {
    ArithmeticException ae = new ArithmeticException("top layer");
    ae.initCause( new IOException("cause") );
    throw ae;
   }
   else
   {
    System.out.println(a/b);
   }
  }
 public static void main(String[] args)
 {
  try {
   divide(5, 0);
  }
```

```
 catch(ArithmeticException ae) { System.out.println(
  "caught : " +ae); System.out.println("actual cause:
  "+ae.getCause());
 }
}
}
```

Sample Output:

caught : java.lang.ArithmeticException: top layer

actual cause: java.io.IOException: cause

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