Chained Exception in Java

Chained Exception was added to Java in JDK 1.4. This feature allows you 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.

Two new constructors and two new methods were added to **Throwable** class to support chained exception.

1. **Throwable**(Throwable cause)
2. **Throwable**(String str, Throwable cause)

In the first constructor, the paramter **cause** specifies the actual cause of exception. In the second form, it allows us to add an exception description in string form with the actual cause of exception.

getCause() and **initCause()** are the two methods added to **Throwable** class.

- **getCause()** method returns the actual cause associated with current exception.
- **initCause()** set an underlying cause(exception) with invoking exception.

Time for an Example!

Lets understand the chain exception with the help of an example, here, ArithmeticException was thrown by the program but the real cause of exception was IOException. We set the cause of exception using initCause() method.

```
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)
```

Output:

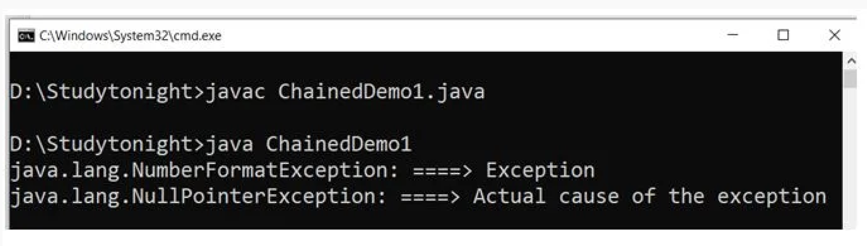
```
caught:java.lang.ArithmeticException: top layer
actual cause: java.io.IOException: cause
```

Example

lets see one more example to understand chain exception, here NumberFormatException was thrown but the actual cause of exception was a null pointer exception.

```
public class ChainedDemo1
{
  public static void main(String[] args)
  {
    try
    {

      NumberFormatException a = new NumberFormatException("====> Exception");

      a.initCause(new NullPointerException("====> Actual cause of the exception"));

      throw a;
    }

    catch(NumberFormatException a)
    {

      System.out.println(a);
```

Output:



Exception propagation

In Java, an exception is thrown from the top of the stack, if the exception is not caught it is put in the bottom of the stack, this process continues until it reaches to the bottom of the stack and caught. It is known as exception propagation. By default, an unchecked exception is forwarded in the called chain.

Example

In this example, an exception occurred in method a1 which is called by method a2 and a2 is called by method a3. Method a3() is enclosed in try block to provide the safe guard. We know exception will be thrown be method a1 but handled in method a3(). This is called exception propagation.

```
class ExpDemo1{
  void a1()
  {
    int data = 30 / 0;
  }
  void a2()
  {
    a1();
  }
  void a3()
  {
    try {
      a2();
    }
    catch (Exception e)
    {
      System.out.println(e);
    }
```

Output:

```
java.lang.ArithmeticException: / by zero
```

Example

Lets take another example, here program throw an IOException from method m1() which is called inside the n1(). Exception thrown by method m1() is handled by method n1().

```
import java.io.IOException;
class Demo{
  void m1() throws IOException
  {
    throw new IOException("device error");
  }
  void n1() throws IOException
  {
    m1();
  }
  void p1()
  {
    try {
      n1();
    }
    catch (Exception e)
    {
      System.out.println("Exception handled");
```

← Prev

Next →