# TheServerSide.com

## New Java 7 Features: Suppressed Exceptions and Try With Resources from Project Coin

One of the great things about the new *try-with-resources* construct is the fact that any AutoCloseable object that is initialized in the resource declaration block will be automatically closed after the code block executes, regardless of whether an exception is thrown, or whether execution proceeds without any problems.

### Use try-with-resources

Given this knowledge and understanding of the try-with-resources construct, deciphering the output of the following Java application is elementary:

```
public class TryWithResources {

        public static void main(String[] args) throws Exception {

                try ( OpenDoor door = new OpenDoor() ) {
                        door.swing(); /*this throws a SwingExecption*/
                }
                catch (Exception e) {
                        System.out.println("Is there a draft? " + e.getClass());
                }
                finally {
                        System.out.println("I'm putting a sweater on, regardless. ");
                }
        }
}

class OpenException extends Exception {}
class SwingException extends Exception {}
class CloseException extends Exception {}

class OpenDoor implements AutoCloseable {

        public OpenDoor() throws Exception {
                System.out.println("The door is open.");
        }
        public void swing() throws Exception {
                System.out.println("The door is becoming unhinged.");
                throw new SwingException();
        }

        public void close() throws Exception {
                System.out.println("The door is closed.");
        }
}
```

When this code runs, we get the following output:

```
The door is open.
The door is becoming unhinged.
The door is closed.
Is there a draft? class SwingException
I'm putting a sweater on, regardless.
```

If you follow the code, you really don't run into anything too interesting or too complex. Basically, the OpenDoor is created in the resource declaration block, the door has its swing method invoked, the swing method throws an Exception, the try-with-resources block calls the close() method on the OpenDoor instance, and then exception handling is performed as expected, catching the SwingException.

### When two exceptions are thrown by the try-with-resources construct

Now, the way Java programs work is that when an exception is thrown, program execution is short-circuited, and control jumps to the first catch block capable of handling that exception. If the exception is handled, program execution resumes from the point in which the exception is handled. If the exception is not handled, the program crashes. So, when our code has the potential to throw an exception, a good developer can write code that handles that exception.

However, the try-with-resources block introduces a new conundrum into the parlance of the Java developer. You see, our swing() method throws an exception, but what would happen if the close() method, which is invoked automatically *before* our error handling block is encountered, throws an Exception as well?

```
class OpenDoor implements AutoCloseable {

        public OpenDoor() throws Exception {
                System.out.println("The door is open.");
        }
        public void swing() throws Exception {
                System.out.println("The door is becoming unhinged.");
                throw new SwingException();
        }

        public void close() throws Exception {
                System.out.println("The door is closed.");
                throw new CloseException(); /* throwing CloseException */
        }
}
```

In this case, we have the bizarre scenario, exclusive to Java 7, where there can be two completely different exceptions thrown before even getting a chance to deal with the first one. It really does beg the question as to what the heck happens in this scenario.

### Handling suppressed exceptions in Java 7

The key to this particular scenario is the "supressed exception." Whenever an exception is thrown within the body, and then followed by an exception thrown by the try-with-resources statement, only the exception thrown in the body of the try is eligible to be caught by the exception handling code. All other exceptions are considered 'supressed exceptions', a concept that is new with Java 7. So,  if we were to run the main method of our TryWithResources class with both the swing() and close() methods throwing exceptions, we would get the same output as when only the swing() class throws an exception, since the CloseException thrown by the close() method would be *suppressed*.

## New methods of java.lang.Throwable in Java 7

However, just because the CloseException is supressed in this scenario, it doesn't mean this suppressed exception has be ignored. To address this concept of suppressed exceptions, two new methods and a constructor have been added to the java.lang.Throwable class in Java 7.

**public final void addSuppressed(Throwable exception)**

Appends the specified exception to the exceptions that were suppressed in order to deliver this exception.

**public final Throwable[] getSuppressed()**

Returns an array containing all of the exceptions that were suppressed, typically by the try-with-resources statement, in order to deliver this exception.

**protected Throwable(String message, Throwable cause,
                    boolean enableSuppression, boolean writableStackTrace)**

Note that this constructor is duplicated in the java.lang.Exception class in Java 7, due the fact that constructors are not inherited by subclasses.

Other constructors of Throwable treat suppression as being enabled. Subclasses of Throwable should document any conditions under which suppression is disabled. Disabling of suppression should only occur in exceptional circumstances where special requirements exist, such as a virtual machine reusing exception objects under low-memory situations. Circumstances where a given exception object is repeatedly caught and rethrown, such as to implement control flow between two sub-systems, is another situation where immutable throwable objects would be appropriate.

## Getting suppressed exceptions in a catch block

So, if we wanted to actually look at the suppressed exceptions, we could adjust the catch block in our main method to print out the suppressed exceptions:

```
public class TryWithResources {

        public static void main(String[] args) throws Exception {

                try ( OpenDoor door = new OpenDoor() ) {
                        door.swing(); /*this throws a SwingExecption*/
                }
                catch (Exception e) {
                        System.out.println("Is there a draft? " + e.getClass());
                        int suppressedCount = e.getSuppressed().length;
                        for (int i=0; i#60;suppressedCount; i++){
                                System.out.println("Suppressed: " + e.getSuppressed()[i]);
                        }
                }
                finally {
                        System.out.println("I'm putting a sweater on, regardless. ");
                }
        }
}
```

And running this code with both the swing() and close() methods throwing their corresponding exceptions, we would get the following output:

```
The door is open.
The door is becoming unhinged.
The door is closed.
Is there a draft? class SwingException
Suppressed: CloseException
I'm putting a sweater on, regardless.
```

As you can see, only the SwingException is handled by the catch, as this was the exception thrown within the body of the try block, and the CloseException is only accessible through the getSuppressed() method of the exception being handled.

*The following is the full Java code, saved in a file named TryWithResources.java, that generates the final output of this tutorial:*

```
public class TryWithResources {

        public static void main(String[] args) throws Exception {

                try ( OpenDoor door = new OpenDoor() ) {
                        door.swing(); //this throws a SwingExecption
                }
                catch (Exception e) {
                        System.out.println("Is there a draft? " + e.getClass());
                        int suppressedCount = e.getSuppressed().length;
                        for (int i=0; i<suppressedCount; i++){
                                System.out.println("Suppressed: " + e.getSuppressed()[i]);
                        }
                }
                finally {
                        System.out.println("I'm putting a sweater on, regardless. ");
                }
        }
}

class OpenException extends Exception {}
class SwingException extends Exception {}
class CloseException extends Exception {}

class OpenDoor implements AutoCloseable {

        public OpenDoor() throws Exception {
                System.out.println("The door is open.");
        }
        public void swing() throws Exception {
                System.out.println("The door is becoming unhinged.");
                throw new SwingException();
        }

        public void close() throws Exception {
                System.out.println("The door is closed.");
                throw new CloseException(); // throwing CloseException
        }
}
```

**Check out these other tutorials from TheServerSide's Sal Pece and Cameron McKenzie covering the new Java 7 features:**

New Java 7 Features: Binary Notation and Literal Variable Initialization
New Java 7 Features: Numeric Underscores with Literals Tutorial
New Java 7 Features: Using String in the Switch Statement Tutorial
New Java 7 Features: The Try-with-resources Language Enhancement Tutorial
New Java 7 Features: Automatic Resource Management (ARM) and the AutoCloseable Interface Tutorial
New Java 7 Features: Suppressed Exceptions and Try-with-resources Tutorial
Java 7 Mock Certification Exam: A Tricky OCPJP Question about ARM and Try-with-Resources
OCAJP Exam to Debuts in March 2010. OCPJP Released in June?
OCPJP & OCAJP Java 7 Exams: Oracle Drops the Training Requirement
OCAJP and OCPJP Changes for Java 7:  New Objectives, a Format Change and a Price Hike

*12 Jan 2012*