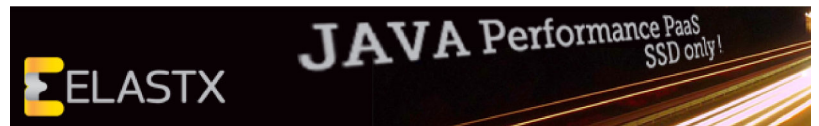




Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER


[Java](#) | [Android](#) | [JVM Languages](#) | [Software Development](#) | [Agile](#) | [DevOps](#) | [Communications](#) | [Career](#) | [Misc](#) | [Meta JCG](#)
 Search this site

You are here: [Home](#) | [Core Java](#) | [Java 7: try-with-resources explained](#)



About **Mohamed Sanalla**



Java 7: try-with-resources explained

by Mohamed Sanalla on July 13th, 2011 | Filed in: [Core Java](#) Tags: [Java 7](#)



This article examines the use of the [try-with-resources](#) statement. This is a try statement that declares one or more resources. A resource is as an object that must be closed after the program is finished with it.

The [try-with-resources](#) statement ensures that each resource is closed at the end of the statement. Any object that implements the `java.lang.AutoCloseable` or `java.io.Closeable` interface can be used as a resource.

Prior to [try-with-resources](#) (before Java 7) while dealing with SQL Statement or ResultSet or Connection objects or other IO objects one had to explicitly close the resource. So one would write something like:

```
01 try{
02     //Create a resource- R
03 } catch(SomeException e){
04     //Handle the exception
05 } finally{
06     //if resource R is not null then
07     try{
08         //close the resource
09     }
10     catch(SomeOtherException ex){
11     }
12 }
```

We have to explicitly close the resource and thereby add a few more lines of code. There are few cases where the developer will forget to close the resource. So to overcome these and other issues – [try-with-resources](#) is introduced in Java 7.

Lets look at an example of how we would use try..catch...finally in pre Java 7. Let me create 2 custom exceptions – ExceptionA and ExceptionB. These will be used through out the examples.

```
1 public class ExceptionA extends Exception{
2     public ExceptionA(String message){
3         super(message);
4     }
5 }
```

```
1 public class ExceptionB extends Exception{
2     public ExceptionB(String message){
3         super(message);
4     }
5 }
```

Lets create some resource, say OldResource, which has two methods – `doSomeWork()`: which does some work and `close()`: which does the closing. Note that this depicts the use of a generic resource – do some work and then close the resource. Now each of these operations, `doSomeWork` and `close`, throws an exception.

```
01 public class OldResource{
02     public void doSomeWork(String work) throws ExceptionA{
03         System.out.println("Doing: "+work);
04         throw new ExceptionA("Exception occurred while doing work");
05     }
06     public void close() throws ExceptionB{
07         System.out.println("Closing the resource");
08         throw new ExceptionB("Exception occurred while closing");
09     }
10 }
```

Lets use this resource in a sample program:

Core Java Job Openings

[www.shine.com/...](#)
Top Jobs For
Experienced
Candidates Register
& Apply Now on
shine.com !

AdChoices

Java

[www.Qui](#)
Multiple Pc
Open In
Desired Fiel
Now. Fr

AdC

DigitalOcean

SSD Virtual Se

\$5 /mo. 20GB SSD Disk

GET STARTED

Launch a server in 55 s

zivane.com

70% OFF

Triumph

SHOP NOW >

EASY RETURNS • CASH ON DEL

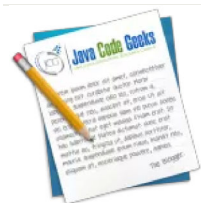
Newsletter



Join our exclusive
news in th
well as ins
Android, S
other rela
As an ext
joining yo

new e-books, published by Jav
their JCG partners for your read

Join Us



With **317**,
visitors ar
authors w
the top Ja
around. C
the looko
encourag

If you have a blog with unique ar
content then you should check c
partners program. You can also

```

01 public class OldTry {
02     public static void main(String[] args) {
03         OldResource res = null;
04         try {
05             res = new OldResource();
06             res.doSomeWork("Writing an article");
07         } catch (Exception e) {
08             System.out.println("Exception Message: "+
09                 e.getMessage()+" Exception Type: "+e.getClass().getName());
10         } finally{
11             try {
12                 res.close();
13             } catch (Exception e) {
14                 System.out.println("Exception Message: "+
15                     e.getMessage()+" Exception Type: "+e.getClass().getName());
16             }
17         }
18     }
19 }

```

The output:

```

1 Doing: Writing an article
2 Exception Message: Exception occurred while doing work Exception Type: javaapplication4.ExceptionA
3 Closing the resource
4 Exception Message: Exception occurred while closing Exception Type: javaapplication4.ExceptionB

```

The program is straight forward: create a new resource, use it, and then try to close it. One can look at the number of extra lines of code there.

Now lets implement the same program using Java 7's try-with-resource construct. For this we would need a new resource – NewResource. In Java 7 a new interface has been, [java.lang.AutoCloseable](#). Those resources which need to be closed implement this interface. All the older IO APIs, socket APIs etc. implement the Closeable interface – which means these resources can be closed. With Java 7, [java.io.Closeable](#) implements [AutoCloseable](#). So everything works without breaking any existing code.

The NewResource code below:

```

01 public class NewResource implements AutoCloseable{
02     String closingMessage;
03
04     public NewResource(String closingMessage) {
05         this.closingMessage = closingMessage;
06     }
07
08     public void doSomeWork(String work) throws ExceptionA{
09         System.out.println(work);
10         throw new ExceptionA("Exception thrown while doing some work");
11     }
12     public void close() throws ExceptionB{
13         System.out.println(closingMessage);
14         throw new ExceptionB("Exception thrown while closing");
15     }
16
17     public void doSomeWork(NewResource res) throws ExceptionA{
18         res.doSomeWork("Wow res getting res to do work");
19     }
20 }

```

Now lets use the NewResource in a sample Program using try-with-resource:

```

01 public class TryWithRes {
02     public static void main(String[] args) {
03         try(NewResource res = new NewResource("Res1 closing")){
04             res.doSomeWork("Listening to podcast");
05         } catch (Exception e){
06             System.out.println("Exception: "+
07                 e.getMessage()+" Thrown by: "+e.getClass().getSimpleName());
08         }
09     }
10 }

```

The output:

```

1 Listening to podcast
2 Res1 closing
3 Exception: Exception thrown while doing some work Thrown by: ExceptionA

```

One thing to note above is that the Exception thrown by the close is being suppressed by the Exception being thrown by the worker method.

So you can right away notice the difference between the two implementations, one using try...catch...finally and the other using try-with-resource. In the example above only one resource is declared as used. One can declare and use multiple resources within the try block, also nest these try-with-resources blocks.

Along with this, a few new methods and a constructor were added to the [java.lang.Throwable](#) class, all related to suppressing the exceptions thrown along with other exception. Best example for this would be- ExceptionA being thrown by the try block will get

for Java Code Geeks and hone addition to utilizing our [revenue](#) monetize your technical writing!

Career Opportunities

Java Developer – Sigma Tech
Baltimore, MD (FULL-TIME) J

Java Developer – Systems Inte
Development Inc. – Washingto
) July 15th, 2013

Java Developer – Sogeti – Nev
TIME) July 15th, 2013

Java Developer – Kobie Marke
(FULL-TIME) July 15th, 2013

JAVA DEVELOPER – Engility C
Diego, CA (FULL-TIME) July

Tags

Akka Android Tutorial Apache Ca
Apache Maven Apache T
Books Cloud Conci
Design Patterns Eclipse
Google GWT Java 7 Java 8 Ja
JavaOne JAXB JBoss JBoss
JPA JSF JSON JUnit
MongoDB NoSQL Ora
Performance Performanc
Play Framework Project
RESTful Web Service:
Spring Spring Data
Spring Security Testi

suppressed by the ExceptionB being thrown by the finally (while closing the resource) and this was the behavior pre Java 7.

However, with Java 7, the thrown exception keeps track of the exceptions it suppressed on its way to being caught/handled. So the earlier said example can be restated as follows. The ExceptionB being thrown by the close method gets added to the List of suppressed exception of the ExceptionA which is being thrown by the try block.

Let me explain you nested try-with-resources and Suppressed exceptions with the following examples.

Nested try-with-resources

```
01 public class TryWithRes {
02     public static void main(String[] args) {
03         try(NewResource res = new NewResource("Res1 closing");
04             NewResource res2 = new NewResource("Res2 closing")){
05             try(NewResource nestedRes = new NewResource("Nestedres closing")){
06                 nestedRes.doSomeWork(res2);
07             }
08         } catch(Exception e){
09             System.out.println("Exception: "+
10 e.getMessage()+" Thrown by: "+e.getClass().getSimpleName());
11         }
12     }
13 }
14 }
```

The output for the above would be:

```
1 Wow res getting res to do work
2 Nestedres closing
3 Res2 closing
4 Res1 closing
5 Exception: Exception thrown while doing some work Thrown by: ExceptionA
```

Note the order of closing the resources, latest first. Also note that the exception being thrown by each of these close() operations is suppressed.

Lets see how we can retrieve the suppressed exceptions:

Suppressed Exceptions

```
01 public class TryWithRes {
02     public static void main(String[] args) {
03         try(NewResource res = new NewResource("Res1 closing");
04             NewResource res2 = new NewResource("Res2 closing")){
05             try(NewResource nestedRes = new NewResource("Nestedres closing")){
06                 nestedRes.doSomeWork(res2);
07             }
08         } catch(Exception e){
09             System.out.println("Exception: "+
10 e.getMessage()+" Thrown by: "+e.getClass().getSimpleName());
11             if (e.getSuppressed() != null){
12                 for (Throwable t : e.getSuppressed()){
13                     System.out.println(t.getMessage()+
14 " Class: "+t.getClass().getSimpleName());
15                 }
16             }
17         }
18     }
19 }
20 }
```

The output for the above code:

```
1 Wow res getting res to do work
2 Nestedres closing
3 Res2 closing
4 Res1 closing
5 Exception: Exception thrown while doing some work Thrown by: ExceptionA
6 Exception thrown while closing Class: ExceptionB
7 Exception thrown while closing Class: ExceptionB
8 Exception thrown while closing Class: ExceptionB
```

The getSuppressed() method is used to retrieve the exceptions suppressed by the thrown exception. Also a new constructor has been added to Throwable class which can be used to enable or disable the suppression of exceptions. If disabled none of the suppressed exceptions are tracked.

Reference: [Java 7 Project Coin: try-with-resources explained with examples](#) from our JCG partner Mohamed Sanaula at the [Experiences Unlimited blog](#).

Related Articles :

- [A glimpse at Java 7 MethodHandle and its usage](#)
- [Design Patterns in the JDK](#)

- [Understanding and Extending Java ClassLoader](#)
- [Java Memory Model – Quick overview and things to notice](#)

You might also like:

- [Inferred exceptions in Java](#)
- [Java 7 try-with-resources](#)
- [Introduction to Default Methods \(Defender Methods\) in Java 8](#)
- [Java Annotations: Explored & Explained](#)
- [Java 7's Support for Suppressed Exceptions](#)



Share and enjoy!

2 Responses to "Java 7: try-with-resources explained"



Alejandro Hdez. Angeles

November 9th, 2012 at 6:20 pm

Excellent tutorial

[Reply](#)



best java training in chennai

July 19th, 2013 at 5:15 pm

Excellent Article on try with resources

[Reply](#)

Leave a Reply

Name (Required)

Mail (will not be published) (Required)

Website

× 3 = twelve

☐ Notify me of followup comments via e-mail

☒ Sign me up for the newsletter!

Submit Comment

Knowledge Base	Partners	Hall Of Fame	About Java Code Geeks
Examples	Mkyong	“Android Full Application Tutorial” series	JCGs (Java Code Geeks) is an independent online community creating the ultimate Java to Java developers resource center. We have a technical architect, technical team lead (senior developer), plenty of junior developers alike. JCGs serve the Java, SOA, Agile and other communities with daily news written by domain experts, article announcements, code snippets and open source projects.
Resources	The Code Geeks Network	GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial	
Software		Android Game Development Tutorials	
Tutorials		Android Google Maps Tutorial	
		Android Location Based Services Application – GPS location	
	Web Code Geeks	Funny Source Code Comments	License Java Code Geeks content is offered under the Creative Commons Attribution-ShareAlike 3.0 Unported License . If you redistribute this content, you must make clear to others the license terms and conditions. The best way to do this is with a link to this web page. Any of the above can be waived if you get written permission from Java Code Geeks. This license impairs or restricts the author's moral rights.
		Java Best Practices – Vector vs ArrayList vs HashSet	
		Android JSON Parsing with Gson Tutorial	
		Android Quick Preferences Tutorial	

© 2010-2012 Java Code Geeks. Licenced under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
All trademarks and registered trademarks appearing on Java Code Geeks are the property of their respective owners.
Java is a trademark or registered trademark of Oracle Corporation in the United States and other countries.
Java Code Geeks is not connected to Oracle Corporation and is not sponsored by Oracle Corporation.