ABOUT ANDY BECK

Andy is a Senior Software Engineer that has sixteen years of experience working on web, desktop and mobile applications. He holds a Bachelor and Master's degree in Computer Science from the University of Virginia and George Washington University respectively. He specializes in working with Java web services and has significant experience working web applications, databases and continuous integration and deployments. He is currently working as a technical lead at a financial technology organization where he supports mobile application services in Java.

Tomcat vs. Jetty vs. Undertow: Comparison of Spring Boot Embedded Servlet Containers

Posted by: Andy Beck in spring January 26th, 2017 0 24823 Views

With the rise in popularity of micro services we have seen a similar rise in popularity of applications with embedded servlet containers. Spring boot is a Java based framework that supports application services. It runs as a standalone jar with an embedded servlet container or as a WAR file inside a container.

Want to master Spring Framework ?

Subscribe to our newsletter and download the Spring Framework Cookbook right now!

In order to help you master the leading and innovative Java framework, we have compiled a kick-ass guide with all its major features and use cases! Besides studying them online you may download the eBook in PDF format!

Download NOW!

In this example, we will focus on the standalone jar with embedded servlet containers. The framework supports three different types of embedded servlet containers: Tomcat (default), Jetty and Undertow. We will compare the three and look at differences in properties, settings, performance and memory. Keep in mind that this example is analyzing the default configuration. There are many ways to optimize the performance or memory usage including to customize the auto configuration and component scanning.

We used Eclipse Neon, Java 8, Maven 3.3.9, Spring 1.4.3, Tomcat 8.5.6, Jetty 9.3.14 and Undertow 1.3.24.

Table Of Contents

NEWSLETTER

139,921 insiders are already enjoying weekly updates and complimentary whitepapers!
Join them now to gain exclusive access to the latest news in the Java world, as well as insights about Android, Scala, Groovy and other related technologies.

Email address:

Your email address

Receive Java & Developer job alerts in your Area

Sign up

JOIN US

With **1,240,600** monthly unique visitors and over **500** authors we are placed among the top Java related sites around. Constantly being on the lookout for partners; we encourage you to join us. So If you have a blog with unique and interesting content then you should check out our **JCG** partners program. You can also be a **guest writer** for Java Code Geeks and hone your writing skills!

1. Setup Spring Boot Application

We will use Maven to setup a new project in Eclipse with the appropriate dependencies. We will use the starter parent for this example but the dependencies in a production application will likely be altered to streamline, optimize or customize.

1.1 Setup Spring Boot Dependencies

The default embedded servlet container is Tomcat. This version of Spring Web 1.4.3 brings in Tomcat version 8.5.6.

pom.xml

```
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>1.4.3.RELEASE</version>
</parent>

<dependencies>
   <!-- TOMCAT -->
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
   </dependency>
</dependencies>
```

1.2 Setup Spring Boot Main Application and Controllers

To setup the Spring Boot application you include the

```
@SpringBootApplication
```

annotation in your Main class. The

```
@SpringBootApplication
```

annotation brings in

```
@SpringBootConfiguration
```

,

```
@EnableAutoConfiguration
```

and

Application.java

```
@SpringBootApplication
@ConfigurationProperties
public class Application {
public static void main(String[] args) {
   SpringApplication.run(Application.class, args);
}
```

You may choose to eliminate this annotation and add the

```
@SpringBootConfiguration
```

alone or to another class that allows you to customize the configuration. The

```
@ComponentScan
```

will scan your application for items like the

```
@Controller
```

you will need to setup a RESTful service. The following controller will return a simple "Hello World" string from a HTTP GET request. We have also included in the bundled example another endpoint mapping that returns a complex object type.

SampleController.java

```
@Controller
public class SampleController {

@Autowired
private ResourceLoader resourceLoader;

@RequestMapping("/")
@ResponseBody
public String home() {
   return "Hello World!";
}
```

1.3 Key Configuration Parameters

The default properties for all the embedded servlet containers are the same. Some of the most important properties to consider are the properties for configuring startup information like ports and application name, TSL, access logs, compression and many more.

For example, to configure SSL add the following to key value pairs to the application.properties.

application.properties

1.4 How to Find Additional Parameters

To explore the parameters for Spring boot applications you can add the Spring actuator dependency and the

```
@ConfigurationProperties
```

annotation to your Main class. You then visit the

```
/configprops
```

endpoint on your application to get a list of the available properties.

Application.java

```
@SpringBootApplication
@ConfigurationProperties
public class Application {
```

pom.xml

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

```
http://localhost:8080/jcg/service/configprops
```

1.5 Change version of Embedded Servlet Containers

The embedded servlet container versions are defined in the following parent dependency from the pom. You can change the version of the embedded servlet container by explicitly including the dependency and identifying a new version in the pom. We will show you how in the examples below.

pom.xml

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-dependencies</artifactId>
   <version>1.3.7.RELEASE</version>
</dependency>
```

2. Tomcat

As Tomcat is the default embedded servlet container there is nothing you need to do to the default implementation to use Tomcat. You can

or

```
application.properties
```

files.

2.2 Change Version of Tomcat

pom.xml

```
<properties><tomcat.version>8.5.6</tomcat.version></properties>

<dependency>
   <groupId>org.apache.tomcat.embed</groupId>
   <artifactId>tomcat-embed-core</artifactId>
   <version>${tomcat.version}</version>
</dependency>
<dependency>
   <groupId>org.apache.tomcat.embed</groupId>
   <artifactId>tomcat-embed-el</artifactId>
   <version>${tomcat.version}</version>
</dependency>
<dependency>
   <groupId>org.apache.tomcat.embed</groupId>
   <artifactId>tomcat-embed-websocket</artifactId>
   <version>${tomcat.version}</version>
</dependency>
```

3. Jetty

To change the embedded servlet container to Jetty you need to edit the pom file to remove the Tomcat dependency and add Jetty.

3.1 Change to Jetty (version 9.3.14)

pom.xml

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
   <exclusions>
      <exclusion>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-tomcat</artifactId>
```

```
14  </dependency>
```

4. Undertow

To change the embedded servlet container to Undertow you need to edit the pom file to remove the Tomcat dependency and add Undertow.

4.1 Change to Undertow (version 1.3.24 final)

Notice the undertow version included in the spring boot starter is incorrect, referring to 1.3.25. You'll need to change it to 1.3.24.Final for this to work at the time of this article.

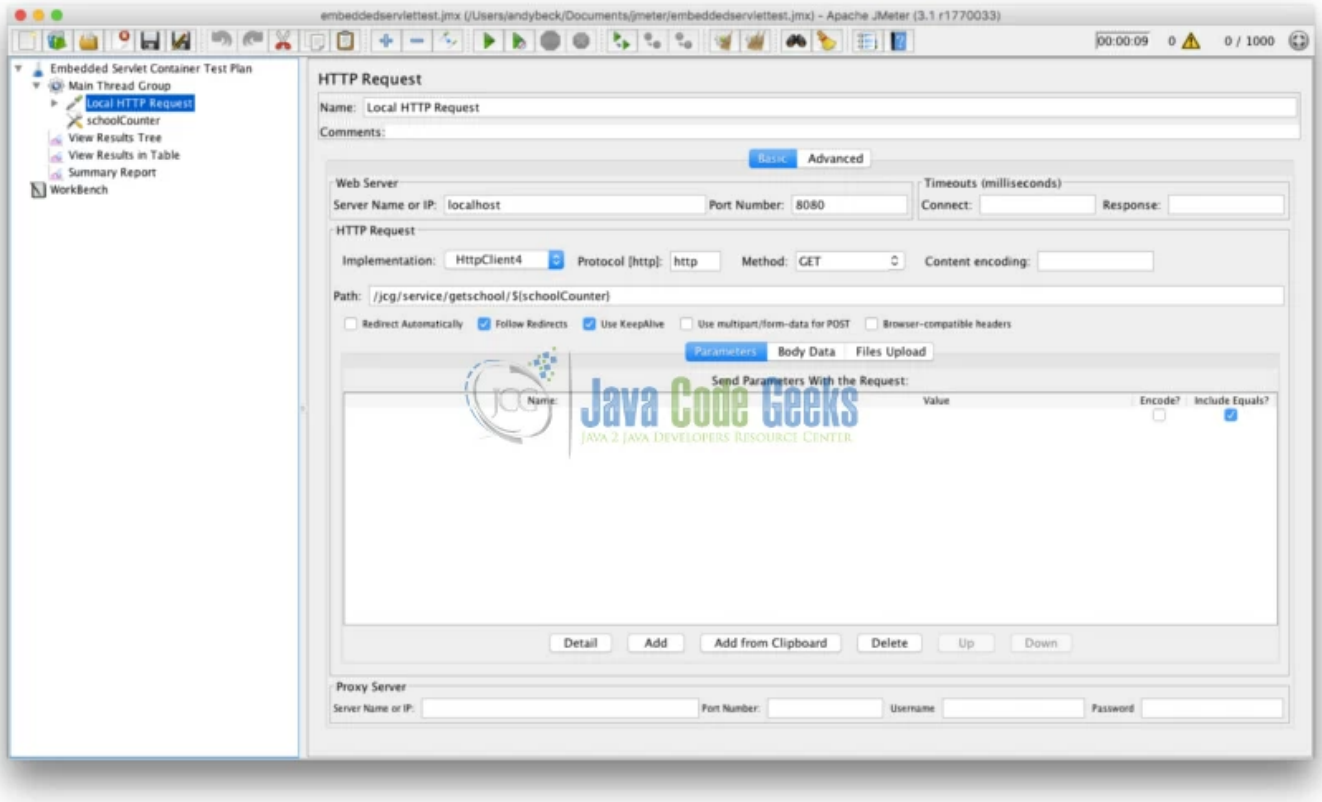
pom.xml

```
01  <dependency>
02     <groupId>org.springframework.boot</groupId>
03     <artifactId>spring-boot-starter-web</artifactId>
04     <exclusions>
05        <exclusion>
06           <groupId>org.springframework.boot</groupId>
07           <artifactId>spring-boot-starter-tomcat</artifactId>
08        </exclusion>
09     </exclusions>
10  </dependency>
11  <dependency>
12     <groupId>org.springframework.boot</groupId>
13     <artifactId>spring-boot-starter-undertow</artifactId>
14  </dependency>
15  <dependency>
16     <groupId>io.undertow</groupId>
17     <artifactId>undertow-core</artifactId>
18     <version>1.3.24.Final</version>
19  </dependency>
20  <dependency>
21     <groupId>io.undertow</groupId>
22     <artifactId>undertow-servlet</artifactId>
23     <version>1.3.24.Final</version>
24  </dependency>
```

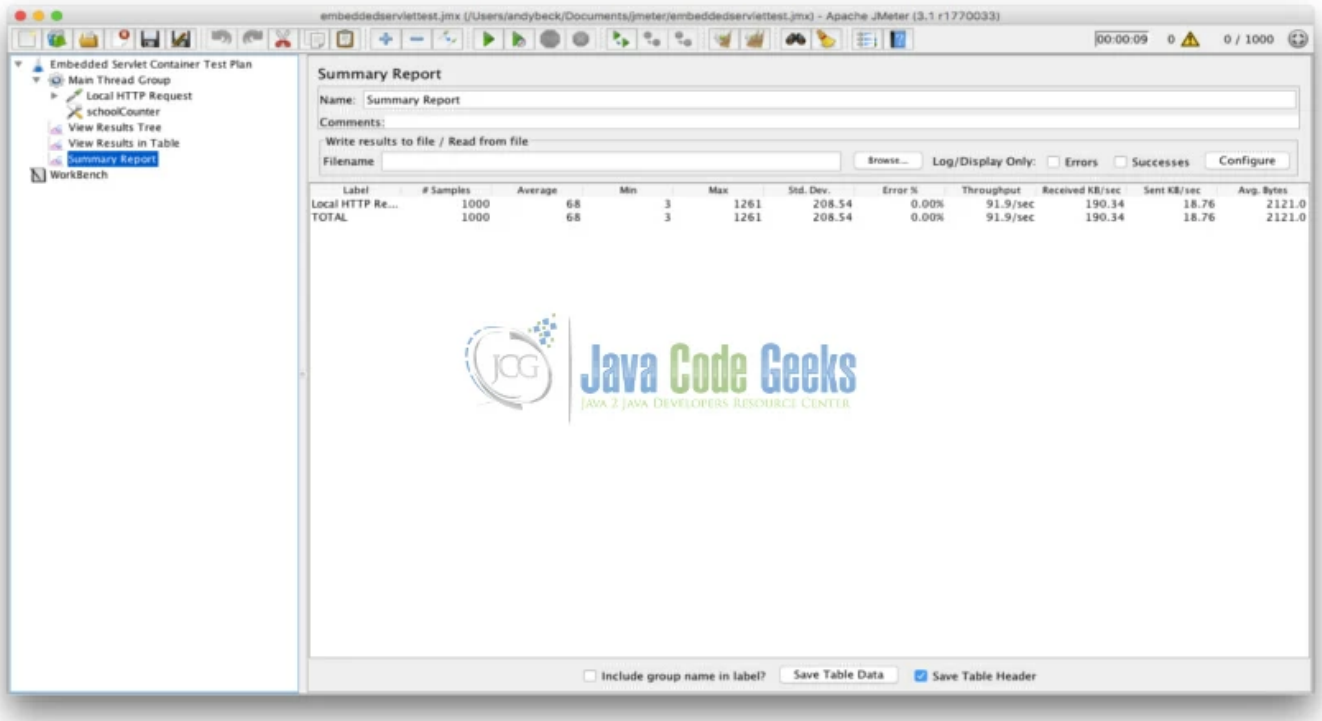
5. Performance and Load

In this example, we will analyze both the peformance of HTTP requests and the memory footprint at startup of all three embedded servlet containers. We used JMeter to measure performance by simulating load and JVisualVM to look at the memory footprint.

report viewers to display or aggregate the results. For the simple string examples, we used a thread group with 1000 threads that would loop 3 times through the sequence. It also used a ramp up time of 10 seconds. For the complex object examples, we used the same parameters but did not loop.



JMeter Tomcat Thread Group



JMeter Tomcat Summary Report

5.1.1 Tomcat

5.1.1.1 Simple String

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Startup	3000	7	1	549	35.78374361	0	293.8583603	55.95935572	55.67238466	195
Others	3000	1	0	45	1.359661682	0	287.8802418	54.82094449	54.53981144	195
Others	3000	1	0	24	1.155032275	0	292.1129503	55.62697785	55.3417113	195

5.1.1.2 Complex Object with Dynamic Data

Others	1000	3	2	17	1.328216473	0	97.88566954	202.7495167	19.9786181	2121
Others	1000	2	1	16	1.110529603	0	98.52216749	204.0678879	20.10852833	2121
Others	1000	2	1	21	1.344498419	0	98.53187506	204.0879951	20.11050966	2121

5.1.2 Jetty

5.1.2.1 Simple Object

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Startup	3000	7	0	561	40.13705065	0	291.5168594	56.0828333	55.22878	197
Others	3000	1	0	21	1.058925031	0	293.5995302	56.48350338	55.6233485	197
Others	3000	1	0	21	0.926034317	0	294.3485086	56.62759395	55.7652448	197

5.1.2.2 Complex Object with Dynamic Data

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Startup	1000	110	3	1397	278.7961107	0	98.13542689	203.3626717	19.93375859	2122
Others	1000	3	2	20	1.500210319	0	98.48335631	204.0836739	20.00443175	2122
Others	1000	3	2	45	2.729377218	0	98.29942003	203.7025091	19.96706969	2122

5.1.3 Undertow

5.1.3.1 Simple Object

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Startup	3000	6	0	451	31.6188702	0	295.6830278	63.81440346	56.01807363	221
Others	3000	1	0	22	1.255447862	0	292.7400468	63.17924839	55.46051669	221
Others	3000	1	0	18	1.559477975	0	294.3773918	63.53262069	55.77071681	221

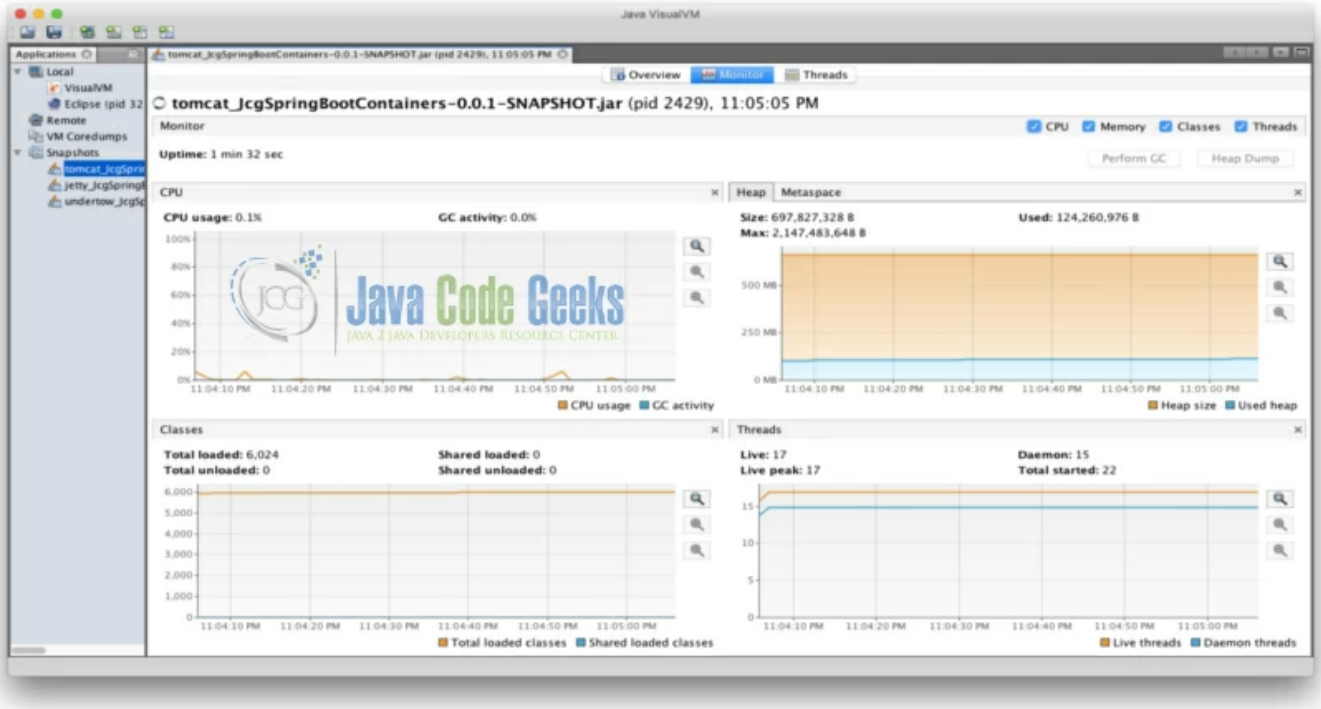
5.1.3.2 Complex Object with Dynamic Data

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes

Others	1000	3	2	19	1.293570253	0	98.55129595	206.6305004	20.01823199	2147
Others	1000	2	2	27	1.659250132	0	98.74592673	207.0385788	20.05776637	2147
Others	1000	2	1	17	1.260904041	0	98.28975821	206.0821395	19.96510714	2147

5.2 Measure Memory

To measure the memory of each embedded servlet container we looked at the memory usage on startup. JVisualVM is a tool provided with the Java Development Kit for visualizing the memory and footprint of java applications. We used this tool to show the initial startup impact of each of the three embedded servlet containers. The heap size and thread counts are key in analyzing this initial footprint. The ten threads that are common to all three containers include: JMX server connection timeout, RMI Scheduler, RMI TCP Connection (2), RMI TCP Accept, Attach Listener, DestroyJavaVM, Signal Dispatcher, Finalizer and Reference Handler.



Heap Size: 697,827,328 B
Used: 124,260,976 B
Max: 2,147,483,648 B

Threads: 17 Live, 22 Started

5.2.3 Jetty

Heap Size: 628,621,312 B
Used: 311,476,776 B
Max: 2,147,483,648 B

Threads: 19 Live, 22 Started

5.2.4 Undertow

Heap Size: 630,718,464 B
Used: 114,599,536 B
Max: 2,147,483,648 B

Threads: 17 Live, 20 Started

6. Compare

6.1 Performance

While all three of the embedded servlet containers had similar performance under the parameters used in this example, Undertow seems to have the best performance with Tomcat and Jetty close behind. The memory footprint of Jetty on startup was the largest using 311 MB. Tomcat and Undertow had similarly low initial footprints around 120 MB with Undertow coming in the lowest at 114 MB. The key difference in the response headers is that Undertow includes HTTP Persistent connections by default. This header will be used in clients that support persistent connections to optimize performance by reusing connection details.

6.1.1 Tomcat Response Headers

```
1 Content-Type →application/json;charset=UTF-8
2 Date →Mon, 09 Jan 2017 02:23:26 GMT
3 Transfer-Encoding →chunked
4 X-Application-Context →JcgSpringBootContainers:# Application index.
```

```
3 Transfer-Encoding →chunked
4 X-Application-Context →JcgSpringBootContainers:# Application index.
```

6.1.3 Undertow Response Headers

```
1 Connection →keep-alive
2 Content-Type →application/json;charset=UTF-8
3 Date →Mon, 09 Jan 2017 02:20:25 GMT
4 Transfer-Encoding →chunked
5 X-Application-Context →JcgSpringBootContainers:# Application index.
```

7. Conclusion

The numbers indicate that Undertow is the best in performance and memory usage. It is encouraging to see that Undertow is embracing the latest capabilities and defaulting to persistent connections. The numbers do not indicate a dramatic difference in performance based on the load used in this example but I would imagine that they would scale and that if performance is the most important factor Undertow is the right match for your application. It is also reasonable to think that an organization may favor an embedded servlet container because of familiarity with it's capabilities. Many times the defaults settings will have to change because of application requirements that include performance, memory usage and functionality.

8. Download the Source Code

Here we compared three types of embedded servlet containers you can include in a Spring Boot Application.

Download
You can download the Eclipse project here: **JcgSpringBootContainers**

Do you want to know how to develop your skillset to become a Java Rockstar?

Subscribe to our newsletter to start Rocking right now!
To get you started we give you our best selling eBooks for FREE!

1. JPA Mini Book
2. JVM Troubleshooting Guide

and many more ....

Email address:

Your email address

Receive Java & Developer job alerts in your Area

Sign up

LIKE THIS ARTICLE? READ MORE FROM JAVA CODE GEEKS

Duplicate Profiles woodworking

Ad Smart Saker

Spring Boot Microservices , Docker and Kubernetes…

javacodegeeks.com

5G: 2020's Top Investment?

Ad Breakthrough Investor

Spring Boot Remove Embedded Tomcat Server, Enable Jetty Server

javacodegeeks.com

Online Selenium Training Madhu - Learn From Industry Leaders

Ad getsoftwareservice.com

Deploying a simple Spring Boot application on Docker | Examples Java…

javacodegeeks.com

Spring Framework Cookbook

javacodegeeks.com

Deploy a Spr Application i Tomcat | Jav Geeks

javacodegeeks.com

Start the discussion...

This site uses Akismet to reduce spam. Learn how your comment data is processed.

Subscribe

KNOWLEDGE BASE

- Courses
- Minibooks
- News
- Resources
- Tutorials

THE CODE GEEKS NETWORK

- .NET Code Geeks
- Java Code Geeks
- System Code Geeks
- Web Code Geeks

HALL OF FAME

- Android Alert Dialog Example
- Android OnClickListener Example
- How to convert Character to String and a String to Character Array in Java
- Java Inheritance example
- Java write to File Example
- java.io.FileNotFoundException – How to solve File Not Found Exception
- java.lang.arrayindexoutofboundsexception – How to handle Array Index Out Of Bounds Exception
- java.lang.NoClassDefFoundError – How to solve No Class Def Found Error
- JSON Example With Jersey + Jackson
- Spring JdbcTemplate Example

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike. JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

DISCLAIMER

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. Examples Java Code Geeks is not connected to Oracle Corporation and is not sponsored by Oracle Corporation.